



Version 1.2

User Manual

Elixir Technology Pte Ltd

Web: www.elixirtech.com

Email: support@elixirtech.com

CONTENTS

C H A P T E R 1	INSTALLATION INSTRUCTION	1
	FOR JDK 1.1	1
	FOR JAVA 2 (JDK 1.2)	1
	INSTALLATION ON MACINTOSH	2
C H A P T E R 2	ELIXIR CASE, OVERVIEW	3
	MORE THAN A DIAGRAMMING TOOL	3
	JAVA SUPPORT	3
	UML SUPPORT	3
	BROWSER INTERFACE	3
	DESIGNER'S NOTEBOOK	4
	PRESENTATION	4
	REVERSE ENGINEERING.....	4
	EXTENSIBILITY	4
C H A P T E R 3	GETTING AROUND	5
	IN THIS CHAPTER.....	5
	WHAT TO DO AFTER LAUNCHING ELIXIR CASE ?	5
	WHAT IS A HOME PAGE ?	6
	WHAT IS A DIAGRAM EDITOR ?	8
	WHAT IS A SPECIFICATION EDITOR ?	8
	Specification	9
	Model.....	9
	GOODIES ON THE MENU BAR.....	11
	CASE (previously LOREx2) menu option	11
	Commit.....	11
	Diagram Editor	11
	Specification Editor	12
	Print Setup	12
	Print	13
	Import Classes	14
	Commit and exit	14
	Exit without saving changes	14
	Edit menu option	15
	Undo.....	15
	Redo.....	15
	Select all	15
	Delete Graphics	15
	Plug In menu option.....	15
	Export/Import	15
	Java Code Generator	18
	Statistics	18
	Go menu option	19
	Back	19
	Forward	19
	Home Page	19
	Help menu option	19
	Help.....	19
	About.....	20

GREAT FINDS IN THE TOOLBOX !	21
Selection	22
Connector.....	22
General Objects	22
Bookmark.....	22
Diagram	23
Image	23
Note.....	23
Shortcut	24
THE TOOL BAR IS JUST A CLICK AWAY	24
Bookmark Listing	25
WHAT IS THE STATUS OF ... ?	25
ACROSS THE BOARD	25
HOW TO QUIT FROM ELIXIR CASE ?.....	28
<i>For Windows environment only</i>	28
WHAT IS NEXT ?.....	28
C H A P T E R 4 USE CASE DIAGRAMS	29
IN THIS CHAPTER.....	29
TOOLS USED IN USE CASE DIAGRAMS	29
WHAT IS A USE CASE DIAGRAM ?	30
COMPONENTS OF A USE CASE DIAGRAM	30
<i>Role</i>	30
<i>Use Case</i>	30
<i>External System</i>	31
<i>Activity boundary</i>	31
HOW TO CREATE A USE CASE DIAGRAM ?.....	31
<i>Activity boundary</i>	31
<i>Use Case, Actor and External System</i>	31
Note.....	31
<i>Relationship</i>	32
Note.....	33
WHAT IS ALLOWED FOR USE CASE DIAGRAMS ?	33
WHAT IS NEXT ?.....	34
C H A P T E R 5 OBJECT SEQUENCE DIAGRAM	35
IN THIS CHAPTER.....	35
TOOLS USED IN OBJECT SEQUENCE DIAGRAMS.....	35
WHAT IS AN OBJECT SEQUENCE DIAGRAM ?	36
COMPONENTS OF AN OBJECT SEQUENCE DIAGRAM	36
<i>Role</i>	36
<i>External System</i>	36
<i>Scenario boundary</i>	36
<i>Class</i>	36
<i>Lifeline</i>	37
<i>Message</i>	38
A message to self	39
HOW TO CREATE AN OBJECT SEQUENCE DIAGRAM ?	39
<i>Scenario boundary</i>	39
<i>Create Classes</i>	39
Note.....	39
Note.....	39
<i>Draw Classes</i>	40

Note.....	43
WHAT IS ALLOWED ON AN OBJECT SEQUENCE DIAGRAM ?	43
WHAT IS NEXT ?	43
C H A P T E R 6 STATECHART DIAGRAM	44
IN THIS CHAPTER.....	44
TOOLS USED IN STATECHART DIAGRAMS	44
WHAT IS A STATECHART DIAGRAM ?	45
COMPONENTS OF A STATECHART DIAGRAM	45
<i>Class (Container)</i>	45
<i>State</i>	45
<i>Concurrent State</i>	46
<i>Start State</i>	46
<i>Stop State</i>	46
<i>History State</i>	47
<i>Transition</i>	47
HOW TO CREATE A STATECHART DIAGRAM ?	47
<i>Class Container</i>	47
<i>State Objects</i>	47
Note.....	47
<i>Transitions</i>	48
WHAT IS ALLOWED ON A STATECHART DIAGRAM ?	48
HOW TO USE START STATE, STOP STATE AND HISTORY STATE ?	48
<i>Start State</i>	48
<i>History State</i>	49
<i>Stop State</i>	49
SIMPLIFICATION TECHNIQUES IN TEXTUAL SPECIFICATION	50
WHAT IS NEXT ?	50
C H A P T E R 7 CLASS DIAGRAM	51
IN THIS CHAPTER.....	51
TOOLS USED IN CLASS DIAGRAMS	51
WHAT IS A CLASS DIAGRAM?	52
COMPONENTS OF A CLASS DIAGRAM	52
<i>Class</i>	52
<i>Class (Container)</i>	52
<i>Interfaces</i>	53
<i>Package</i>	53
<i>Relationship</i>	53
Association	53
Extends	54
HOW TO CREATE A CLASS DIAGRAM ?	54
<i>Package</i>	54
<i>Class Container and Class</i>	55
<i>Relationship between classes</i>	55
Extends.....	56
Association	56
<i>Relationship between packages</i>	57
WHAT IS ALLOWED ON A CLASS DIAGRAM ?	57
WHAT IS NEXT ?	57
C H A P T E R 8 THE LIBRARY SYSTEM	58
ANALYSIS	58

OBJECT IDENTIFICATION PHASE	58
<i>Use Case Diagrams</i>	58
Library Loan Policies.....	59
Library Reservation Policies	59
Textual Description of Basic Functions.....	60
Textual Description of Administration	62
<i>Noun Analysis</i>	63
BEHAVIOURAL MODELLING PHASE.....	63
<i>Object Sequence diagrams</i>	63
<i>Pre-conditions and Post-conditions</i>	65
<i>Statechart diagrams</i>	66
MORE OBJECT SEQUENCE DIAGRAMS AND STATECHART DIAGRAMS AFTER CYCLICAL ANALYSIS.....	67
Cancel A Reservation.....	68
STRUCTURE MODELLING PHASE	69
<i>Class Diagrams</i>	69
EXERCISE	70
A P P E N D I X A POPUP MENUS	71
POPUP MENU ITEMS.....	71
<i>Properties</i>	71
<i>Delete Graphic</i>	71
<i>Delete Object</i>	72
<i>Specification</i>	72
<i>Show Connectors</i>	72
<i>Show Contents</i>	72
<i>Add Node</i>	73
<i>Delete Node</i>	73
<i>Add Field</i>	73
<i>Edit Fieldname</i>	73
<i>Add Method</i>	73
<i>Edit Method</i>	73
A P P E N D I X B PROPERTIES	74
THE DIAGRAM PAGE.....	74
<i>Identity</i>	74
<i>Draw Order</i>	75
<i>Note</i>	76
<i>Background Image</i>	76
GENERAL OBJECTS.....	77
<i>Diagram</i>	77
<i>User</i>	77
<i>Image</i>	79
<i>Image</i>	79
<i>Note</i>	79
<i>Note</i>	79
<i>Shortcut</i>	80
<i>Shortcut</i>	80
OTHER PROPERTY TAB PAGES	82
<i>Fields</i>	82
<i>Fields.General</i>	83
<i>General</i>	84
<i>Note</i>	85
<i>Identity (with Activity)</i>	85

<i>Identity (with Parent)</i>	86
<i>Interactions</i>	86
<i>Methods</i>	87
<i>Methods.General</i>	87
<i>Methods.Interactions</i>	89
<i>Show</i>	90
<i>State</i>	91
<i>Transition</i>	92
<i>Trigger</i>	92
<i>Interactions</i>	93

Figure List

Figure 1.1	Elixir CASE Start Screen.....	2
Figure 3.1	Window to Create / Select Home Page	6
Figure 3.2	Elixir CASE.....	6
Figure 3.3	Home Page for a Library System	7
Figure 3.4	The Specification Editor	8
Figure 3.5	The Model tab page	10
Figure 3.6	Elixir CASE Menu Bar	11
Figure 3.7	Print Configuration	12
Figure 3.8	Java 2 Page Setup	13
Figure 3.9	Print window (for Windows platform)	13
Figure 3.10	Import Classes.....	14
Figure 3.11	Export	16
Figure 3.12	Import	17
Figure 3.13	Import Mapper	17
Figure 3.14	Java Code Generator	18
Figure 3.15	Statistics.....	19
Figure 3.16	Online Help.....	20
Figure 3.17	Toolbox	21
Figure 3.18	The Tool Bar.....	24
Figure 3.19	List of Bookmarks	25
Figure 3.20	Choose Object Window	26
Figure 4.1	Tools for Use Case Diagrams	29
Figure 4.2	A Library Use Case Diagram	30
Figure 4.3	Pop-up Menu to Define Relationship Between Use Cases	32
Figure 4.4	"Uses" and "Extends" Relationships.....	33
Figure 5.1	Tools for Object Sequence diagrams	35
Figure 5.2	Fields and Methods available in Class "Book"	37
Figure 5.3	Scenario to Notify Member of Availability of Reserved Book	38
Figure 5.4	Trigger tab page	40
Figure 5.5	Partial "Borrow a Book" OID	41
Figure 5.6	To add further invocations.....	41
Figure 5.7	OID after the invocation of Loan:create(Member, BookCopy).....	42
Figure 5.8	Completed "Borrow a Book" OID.....	42
Figure 6.1	Tools for Statechart diagrams	44
Figure 6.2	Concurrent States	46
Figure 6.3	Start States	48
Figure 6.4	One Distinct Stop State	49
Figure 6.5	State Diagram with Implicit Stop States.....	49
Figure 6.6	The State Transition of Class "Stopwatch" using clauses.....	50
Figure 7.1	Tools for Class Diagrams	51
Figure 7.2	Types of Associations on a Class Diagram.....	54
Figure 7.3	Pop-up Menu of a connector linking class to class	55
Figure 7.4	Extends relationship	56
Figure 7.5	Association between Classes "Reservation" and "BookCopy"	56
Figure 7.6	Cross-package class	57
UCD1	Library System – Policy	59
UCD2	Library System – Administration.....	60
OID1	Borrow a Book – Successful	63
OID2	Return a Book – Overdue	64
OID3	Reserve a Book – Successful.....	64
OID4	Collect a Reservation - Successful	65

STD1	Initial BookCopy Statechart diagram.....	66
STD2	Revised BookCopy Statechart diagram	67
OID5	Copy Available	67
OID6	Cancel a Reservation	68
STD	Reservation Statechart diagram	68
CD1	First-cut Class Diagram.....	69
CD2	Final Class Diagram.....	70
Figure A.1	Popup Menu for a Class.....	71
Figure B.1	Identity tab page	74
Figure B.2(i)	Draw Order tab page	75
Figure B.2(ii)	Corresponding Diagram , showing the object graphics on the page.....	75
Figure B.3	Background tab page	77
Figure B.4(i)	User tab page	78
Figure B.4(ii)	Corresponding UserProperties entry under Specification Editor.....	78
Figure B.5	Image tab page for an Image object graphic	79
Figure B.6	Text tab page for a Note object graphic	80
Figure B.7	Shortcut tab page for the Shortcut object graphic	80
Figure B.8	Fields tab page	82
Figure B.9	Fields' Sub-Properties window upon choosing Add or Edit button	83
Figure B.10	General tab page	84
Figure B.11	Identity tab page (with Activity field).....	85
Figure B.12	Identity tab page (with Parent field).....	86
Figure B.13	Methods tab page	87
Figure B.14	Methods' Sub-properties window upon choosing Add or Edit button	88
Figure B.15	Methods' Interaction tab page	89
Figure B.16(i)	Show tab page.....	90
Figure B.16(ii)	Class "Book" with Selected Fields and Methods and its Stereotype shown...	90
Figure B.17	State tab page	91
Figure B.18	Transition tab page	92
Figure B.19	Trigger tab page	93
Figure B.20(i)	Interactions tab page	94
Figure B.20(ii)	Corresponding Object Sequence diagram	94
Figure B.21	Add Invocation	95

Chapter 1

Installation Instruction

This chapter explains how to install the Elixir CASE software on the Unix, Linux and Windows NT/98/95 platforms. Mac installation is quite different, and is detailed in a later section in this chapter.

Elixir CASE is provided as a jar file. For example, for Elixir CASE version 1.2.4, the full release name is ElixirCASE-1.2.4.jar, while the Lite version is ElixirCASE-1.2.4-Lite.jar.

NOTE :

Please ensure that your system already has an installed Java runtime that supports JDK version 1.1 and above.

For JDK 1.1

First, you'll need to add the ElixirCASE jar file to your class path.

To launch Elixir CASE, type the following at the system prompt:

```
java -mx32m -noclassgc elixir.CASE
```

For Java 2 (JDK 1.2)

No installation is required. To launch Elixir CASE, type the following at the system prompt:

```
java -mx32m -jar ElixirCASE-1.2.4.jar
```

(Use ElixirCASE-1.2.4-Lite.jar if you are installing the Lite version).

For **Windows users**, you may double-click on the jar file to launch the software. (It is recommended that you run from a console the first time, to test your setup, as error messages are lost when you launch from Windows.)

Figure 1.1 Elixir CASE Start Screen

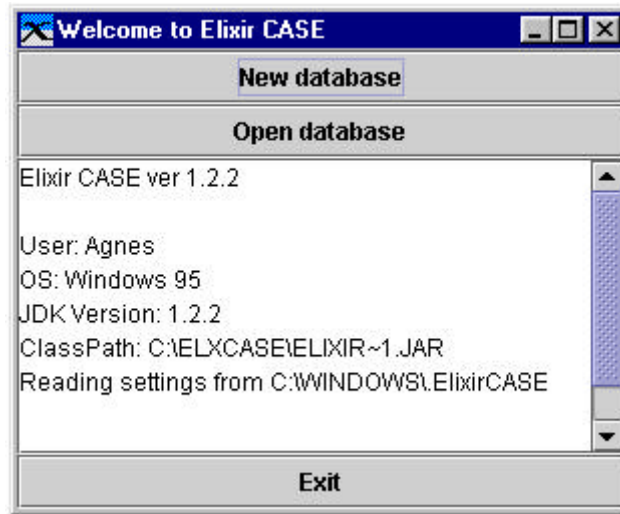


Figure 1.1 above shows the first screen you see when you launch Elixir CASE. From here, you have the option to create a new database or to open an existing database.

Chapter 2 presents an overview of Elixir CASE, and in chapter 3, we will learn how to get around the tool.

Installation on Macintosh

1. Ensure that a recent MRJ is installed.
2. Ensure that Swing is installed. If necessary, download from JavaSoft and install swingall.jar in System Folder/Extensions/MRJLibraries/MRJClasses.
3. Unzip the ElixirCASE zip file with Stuffit or a similar tool. This produces an ElixirCASE jar and a few supporting files.
4. Launch JBindery (an MRJ application).
5. In the Command page of JBindery, fill in the class name: *com.elixirtech.CASE*. Set Redirect stdout: to *Message Window*.
6. In the Classpath page, click *Add .zip File* and choose the ElixirCASE jar file.
7. In the Appearance page, select *Size boxes intrude*.
8. In the Application page, set the Creator Id to ELX1.
9. Choose *Save Settings* and name the file ElixirCASE.
10. Quit JBindery and double-click the ElixirCASE application that it has generated to start Elixir CASE.

Chapter 2

Elixir CASE, Overview

More than a Diagramming Tool

Elixir CASE is more than just a diagramming tool. The main difference is that every graphic on a diagram is a representation of the real data held inside the object database. This makes it possible for a class to be shown on more than one diagram. It offers some obvious benefits, such as, if you were to change the name of a class, all of the graphics will be updated to show the new name. In Elixir CASE, it not only means all graphics of the class will be updated, but that all constructors (which have the same name as the class), and all methods that take a parameter, return a result or throw an exception of that class, will also be consistently maintained. Most CASE tools, even with an object database behind, do not have such a completely unified representation.

Java Support

Elixir CASE is written entirely in Java and is intended solely for Java developers. It does not have a 'common denominator' approach whereby only features supported by all languages are acceptable. In Java you can specify synchronized methods, transient fields and so on, as Elixir CASE captures all of these. Even deprecated methods are identified.

Being written in Java also offers other benefits. You can see what is possible in pure Java and maybe use some ideas for your own systems. Also, we have portability across all Java 1.1 and 1.2 compliant machines. Of course there are a few limitations as a result of our pure Java stance. We are not able to support copying of graphics to the clipboard yet, because it is not operational in Java 1.1. We are ready to add support for this as soon as it becomes possible.

UML Support

Elixir CASE supports the UML notations that we have found essential for Java development: Use Case Diagrams, Sequential Diagrams, Statechart Diagrams, Class and Package Diagrams. While UML contains several more notations, these are occasionally esoteric variations of the mainstream diagrams. Subsequent versions of Elixir CASE will add further diagrams. We welcome customer feedback as to the priority for delivering them.

Browser Interface

Elixir CASE adopts a novel interface for CASE tools: a variation on the familiar browser interface. The first page is referred to as the Home Diagram for the project and new diagrams can be constructed and traversed to through hyperlinks (our diagram icon). The tool thus supports arbitrarily complex networks of diagrams to allow any kind of partitioning of the object model. To make navigation simple, bookmarks and shortcuts are also provided, making leaps between related diagrams a single-click operation.

Designer's Notebook

Rather than restrict, as all other CASE tools do, to a single diagram per page, Elixir CASE allows any combination of pictures – just like a designer's notebook. The tool is also intelligent enough to determine what connector types are appropriate between different graphics, to avoid presenting the user with a slew of arrow variations where the onus is on the user to remember the often inconsistent UML notations for different situations.

Presentation

Following the browser metaphor, you can change the background image of any diagram and insert images easily to form visually appealing presentations to customers. Combined with the shortcut mechanism, it is easy to create slide shows of diagrams to step through a presentation.

Reverse Engineering

Elixir CASE allows the import of Java .class files from the classpath. This reverse engineering extracts a lot of detail from the classes into the object model so that diagrams can be quickly constructed. Elixir CASE offers another first with the generation of sequence diagrams as well as class and package diagrams. Of course, some software licenses prohibit the reverse engineering of codes – Elixir CASE must not be used to breach such license agreements.

Extensibility

Finally, extensibility is a key concern for Elixir CASE. With the ability to define new stereotypes and their icons, and to add user-defined plug-ins which provide more functionalities to the tool, Elixir CASE is limited only by the user's Java programming skills!

Chapter 3

Getting Around

In Chapter 1, we have already seen how to launch Elixir CASE. This chapter explains how to move around Elixir CASE. It introduces the various tools provided in Elixir CASE, and helps us get familiar with the environment and the terms used in Elixir CASE. Readers are encouraged to try out the different tools as we go through the chapter.

The two appendices in the later part of the manual complements this chapter and those after. It would be beneficial for you to refer to them when references are made to them in the course of going through the chapters.

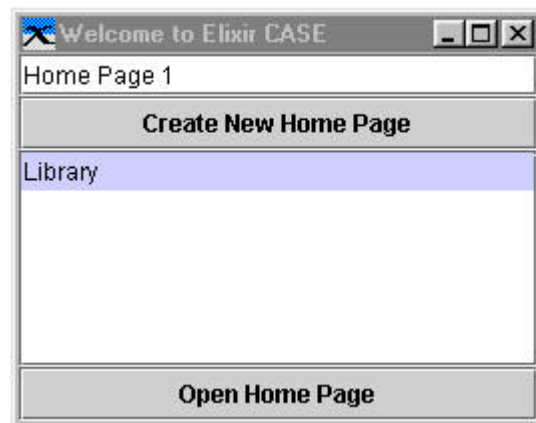
In This Chapter

- ☐ How to Start Up Elixir CASE ?
- ☐ What is a Home Page ?
- ☐ What is a Diagram Editor ?
- ☐ What is a Specification Editor ?
- ☐ Goodies on the Menu Bar
- ☐ Great Finds in the Toolbox !
- ☐ The Tool Bar is just a Click Away
- ☐ What is the Status of ... ?
- ☐ Across the Board
- ☐ How to Quit from Elixir CASE ?
- ☐ What is Next ?

What to do after launching Elixir CASE ?

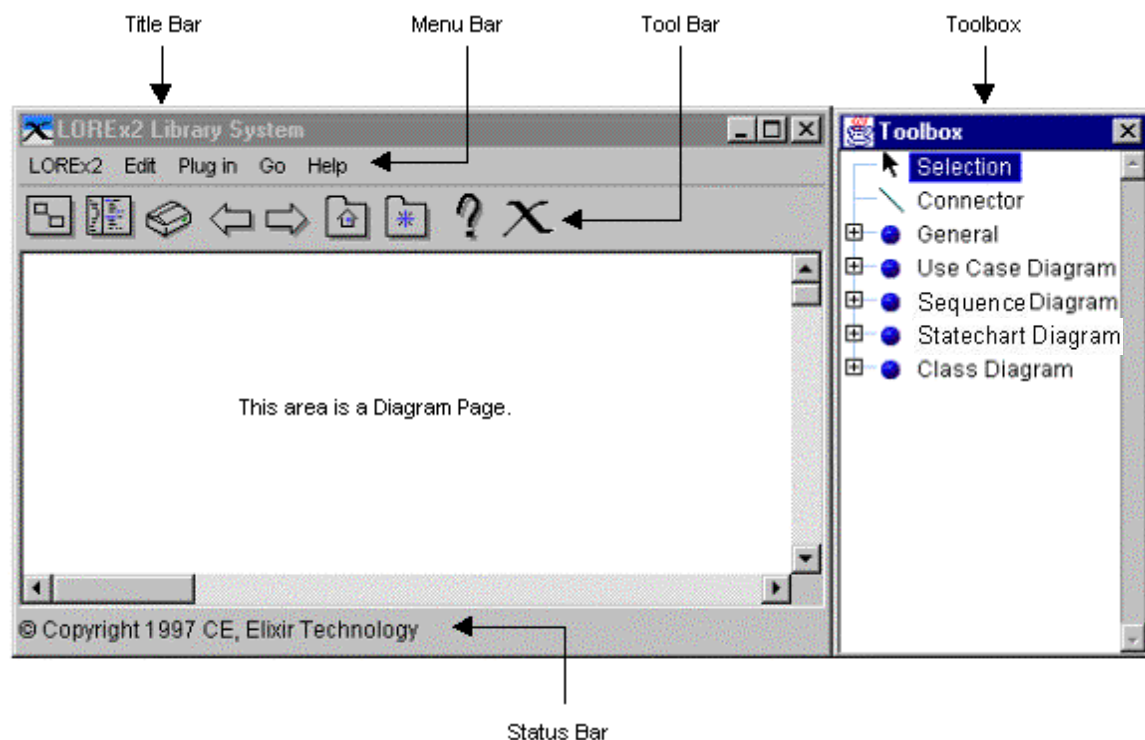
After launching Elixir CASE, you see a welcome screen that allows you to create a new Home Page or to select an existing Home Page. To create a new Home Page, enter in the first field the name of the Home Page and choose the **Create New Home Page** button. In Figure 3.1, the new Home Page would have the name "Home Page 1". We also have an existing Home Page called "Library". To choose this, select "Library" by highlighting it and choosing the **Open Home Page** button.

Figure 3.1 Window to Create / Select Home Page



Then, we will come to the screen as shown in Figure 3.2, where it shows the various tools provided by Elixir CASE.

Figure 3.2 Elixir CASE



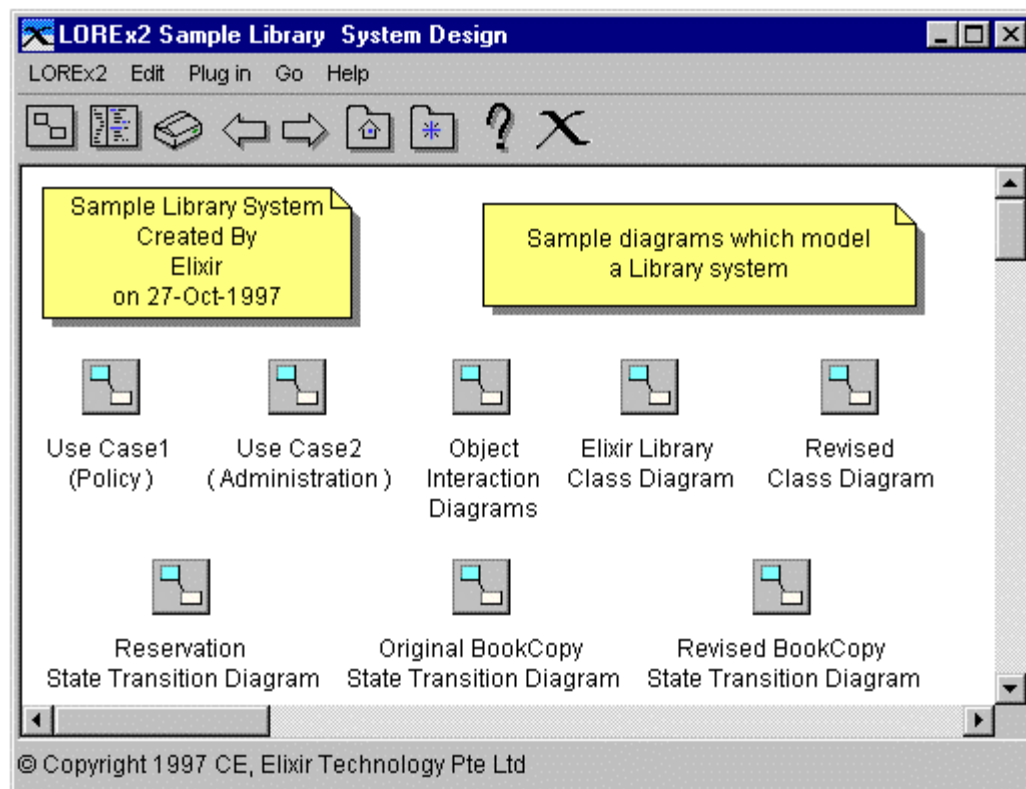
What is a Home Page ?

The Home Page is the first page you see when you start up Elixir CASE. It is identified by a yellow note posted at the top left-hand corner of the page, as in Figure 3.3. The note tells you who created the Home Page and when it was created. You can edit this note.

Basically, the main components of an Elixir CASE screen are the Title Bar, the Menu Bar, The Tool Bar, the Toolbox, the Diagram Page and the Status Bar. The Title Bar tells you which diagram you are on. The Diagram Page is where you draw the diagrams and the Status Bar shows the status of your current processing and also doubles up as the message line. The Menu Bar, Tool Bar and Toolbox are tools to assist us in the drawing of the diagram. We will see them later.

What you see on the screen is just a portion of a page. A page is not just limited to the screen size. You are able to scroll up or down, left or right using the scroll bars. Of course, you are not limited to just one page. You can easily create as many diagram pages as you want and still be able to easily access the pages. In Figure 3.3, what you see are the diagram icons pointing to the various diagram pages that was created. We will see how to create new diagram pages later using the Diagram tool from the Toolbox.

Figure 3.3 Home Page for a Library System



In this way, the Home Page acts as a reference page where you could refer to when you want to go from one page to another, just like when you surf the Internet. Another way of viewing the Home Page is as a root page with many sub-pages, as in an inverted tree with many branches and sub-branches.

Now, let's familiarize ourselves with the environment that we are going to work on !

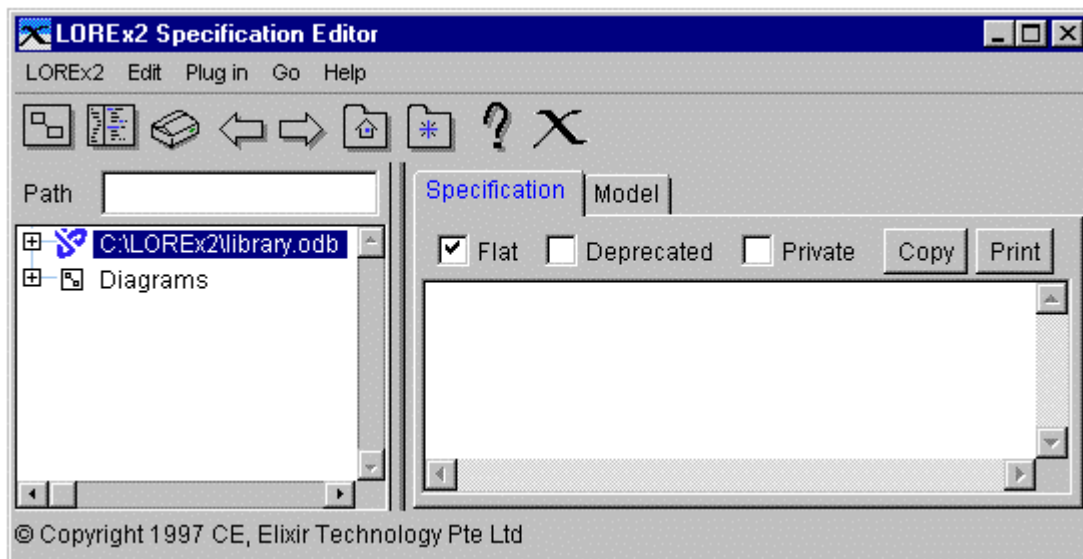
What is a Diagram Editor ?

The Diagram Editor actually refers to the area where you draw and edit your diagrams. The bulk of the space on the Diagram Editor is occupied by the Diagram page. In addition, the Diagram Editor provides the Toolbox with which you work, as shown in Figure 3.2.

What is a Specification Editor ?

The Specification Editor provides you with a directory listing of all the object models that reside in the database and all the objects that are drawn on the diagram page.

Figure 3.4 The Specification Editor



The Specification Editor is separated into 2 adjacent panes.

The **Left Pane** shows the listing of all the objects that have been defined in the current database and their hierarchy. The 'Path' field shows the full path of the currently selected object. If you know the full path of the object that you want to see, you can also enter it in this field to select the object. This field is case-sensitive. Elixir CASE has a standard path format :

FirstObject[.object1.objectn]

where

FirstObject can be a Home Page or Diagram (which points to a page) or an object; this is at level 0.

object1 ... objectn are the objects that are on level 1 ... level n.

',' (the point) is the delimiter to indicate the hierarchy.

There are 2 parts to the hierarchical listing of the objects. The first part has the heading of "C:\Elixir CASE\library.odt", for example [Refer to Figure 3.4]. It lists the actual object models in the database. The second part has the heading "Diagrams". It lists the graphical objects that are drawn on the diagrams.

You can rename an object by right-clicking on it to call up the popup menu and selecting "Properties". Similarly, you can also delete an object or make changes to the properties of the object. Alternatively, you can perform these functions on the graphical objects from the Diagram Editor. The changes made will be effected throughout the whole database and across all diagrams.

The **Right Pane** is linked to the left pane, where it shows the specification and model information of the selected object on the left. The 2 tab pages on this pane are explained below :

Specification

The Specification tab page is as shown in Figure 3.4. It provides the details of the object selected on the left pane. The features are described below :

Fields	Feature	Description
Text	Display Area	Area where the textual specifications are displayed. This area is non-editable.
Flat	Check Box	To choose to display or not to display the flattened inheritance hierarchy [that is, a hierarchy complete with all the inherited features].
Deprecated	Check Box	To include the outdated functions. This is for the purpose of backward compatibility.
Private	Check Box	To include the private features of the selected object.
Copy	Push Button	To copy the whole text in the display area to the clipboard. You can then paste it on any textual part in the Elixir CASE software, such as a Note, or you can paste it to a file outside Elixir CASE, such as a Windows Notepad.
Print	Push Button	To print out the specification in the text field. It will call up the print dialog box for you to choose your settings and then print.

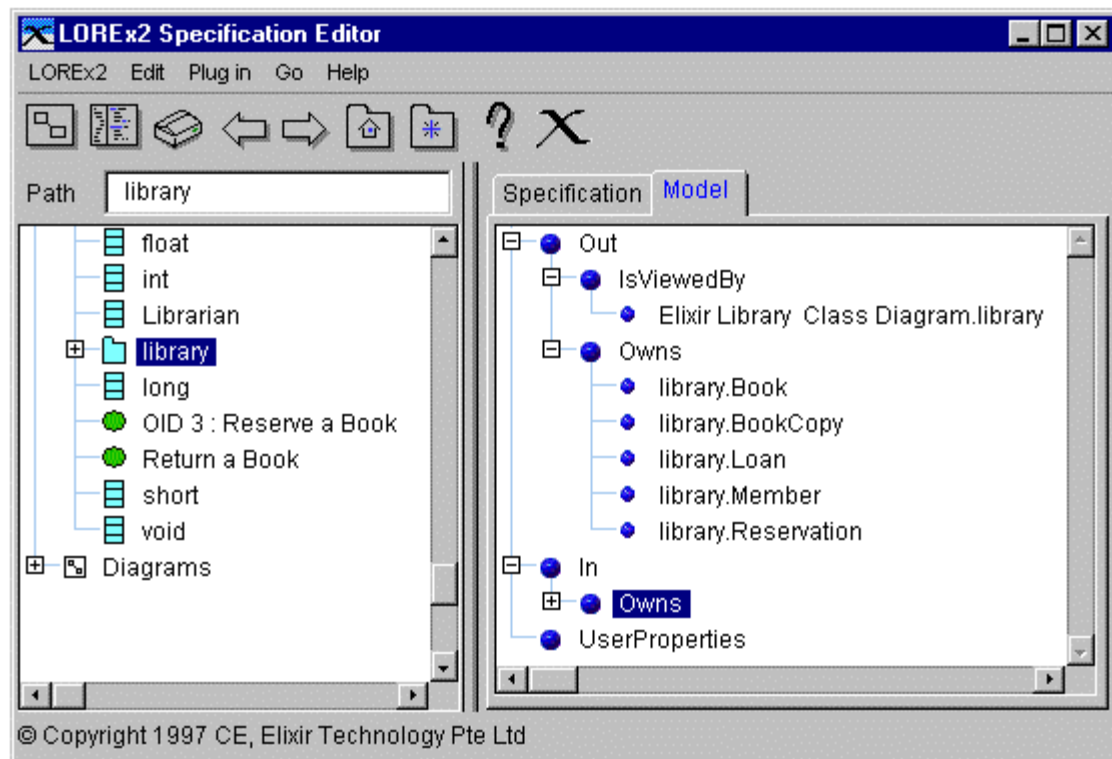
Model

The Model tab page is as shown in Figure 3.5. It shows the relationship of the selected object with other objects. It also shows the user defined properties of the object. They are explained below :

Property	Description
Out	This shows the outward relationship of the object with other objects. Possible relationships are "Owns", "IsViewedBy" and so on.
In	This shows the inward relationship of the object with other objects. Possible relationships are "Owns", "IsViewedBy" and so on.
UserProperties	This shows all the various properties of the object that are defined by the user, such as, version number, date of creation, special

	<p>comments and so on.</p> <p>The properties and their values can be defined on the User tab page of each object [Refer to Appendix B – User tab page].</p> <p>This facility is useful in that the properties defined here can be scanned by the tool and then incorporated into your own codes.</p> <p>You can write your own plug-ins to do that.</p>
--	---

Figure 3.5 The Model tab page



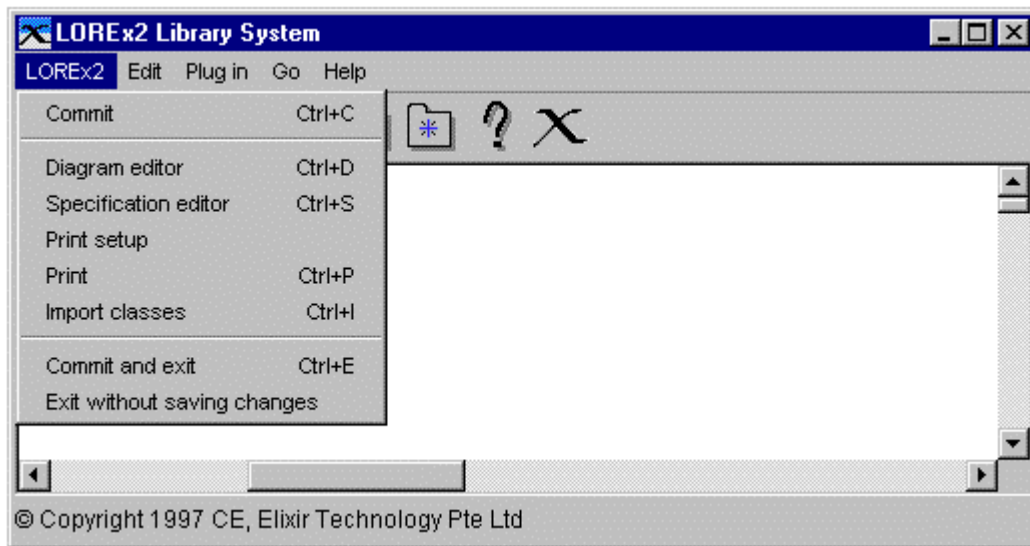
Soon, this environment will be like a second home to you. So, let's learn how to use the various tools that Elixir CASE provides.

Goodies on the Menu Bar

The Menu Bar is available on both the Diagram Editor and Specification Editor. From Figure 3.6, we can see that there are 5 menu options on the menu bar. We will open up each menu option and find out what commands there are.

Elixir CASE provides hotkeys (or shortcut keys) for the commands under the menu options and they are shown on the right of each command.

Figure 3.6 Elixir CASE Menu Bar



CASE (previously LOREx2) menu option

The following commands are available under the Elixir CASE menu option :

Commit

This allows you to commit changes without exiting from Elixir CASE. This should be performed regularly, as it prevents the loss of data in the case of machine failure. It also reduces memory requirement by saving changes to disk.

Diagram Editor

As mentioned above, the Diagram Editor refers to the area where you draw your diagrams. Upon activation of this command, the last accessed diagram page will appear.

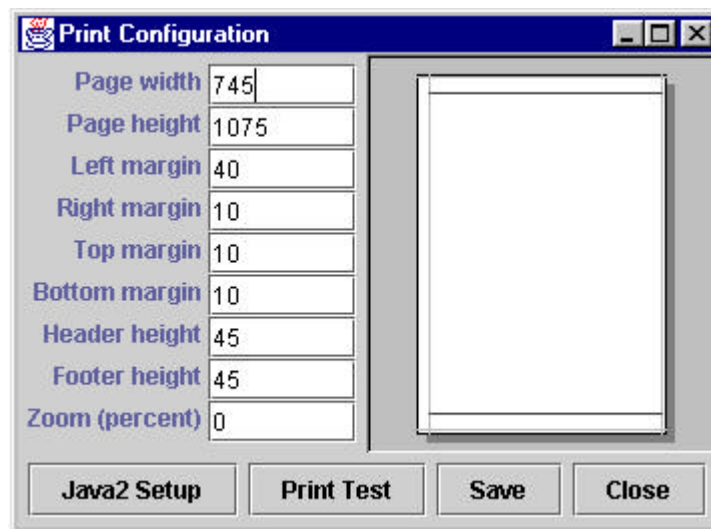
Specification Editor

The Specification Editor provides you with a directory listing of all the objects that exist in the database, as described above. Upon activation of this command, the Specification Editor screen will appear.

Print Setup

The Print Setup allows you to set up the print configuration such as page width and height, margin widths, and header and footer widths, as shown in Figure 3.7. The dimensions are in pixels.

Figure 3.7 Print Configuration



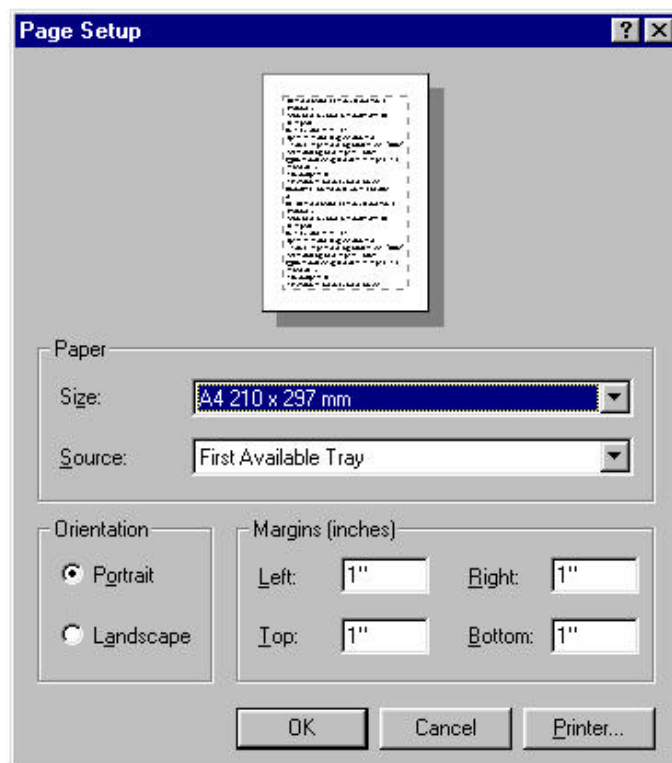
The “Java2 Setup” button is to open the Java 2 page setup dialog, as in Figure 3.8.

The “Print Test” button is for you to do a test print to verify the settings that you have defined. The test printout will be as the page displayed on the screen, showing the lines to delineate the different areas on the page. Due to a limitation in Java, this needs to be done each time a different printer is used.

The “Save” button updates printconfig.ini so that the settings will be remembered for future sessions. Closing the dialog applies the settings to the current session.

The last field is the “Zoom” option. It allows you to produce scaled output, and only works in JDK 1.2 or later. The Zoom value can be set from 0 to 100%, where 0% is reserved to indicate fit-to-page scaling (that is, the diagram is scaled to fit the current page dimensions).

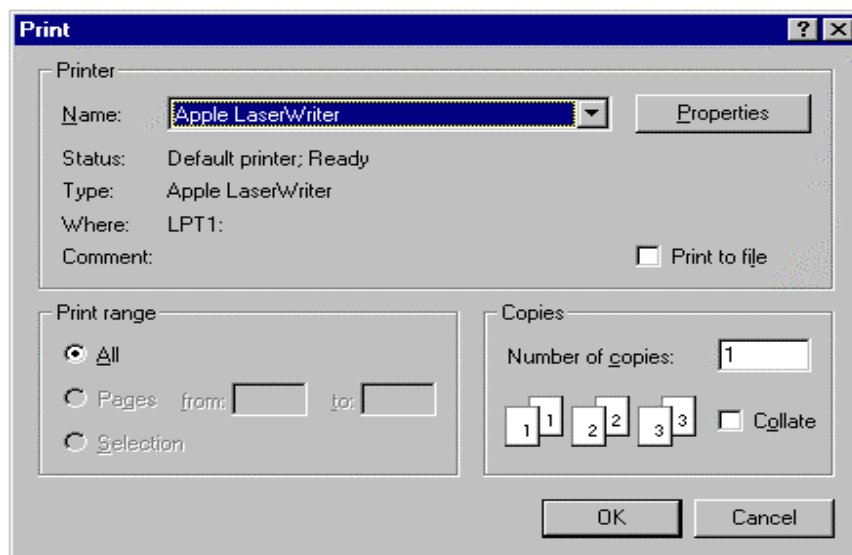
Figure 3.8 Java 2 Page Setup



Print

Activating the “Print” command will call up the Print window. Figure 3.9 shows a sample Print window for the windows platform. Here you can specify the destination printer, the paper size and paper orientation before sending your documents for printing.

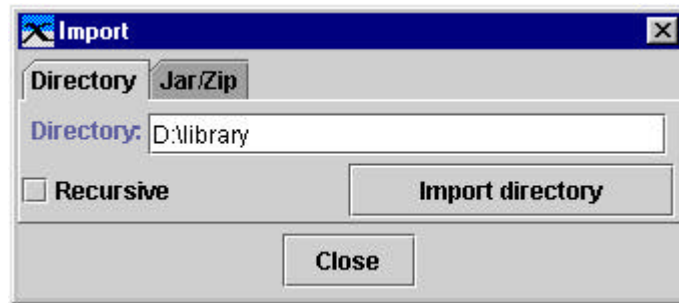
Figure 3.9 Print window (for Windows platform)



Import Classes

This function allows you to select Java classes and reverse engineer them into the UML model. Activating it will call up an Import window as shown in Figure 3.10.

Figure 3.10 Import Classes



The function allows importing of all Java classes in a directory as well as a jar file and zip file. The import is done with a commit, and is not reversible. To remove the classes/packages imported, you are able to delete them.

Classes in one package, for example the library package, will automatically be stored in the database as the package library.

After importing them, you will be able to see them in the Specification Editor. You will also be able to use them in your system.

Note

These classes take up quite a lot of space. Therefore, you may like to remove them if they are not being used. To do so, you can just delete them from the Specification Editor.

Importing can be quite a long operation, and requires considerable memory. To import a large jar or zip file, it is recommended that you extract the files into directories and import the chosen directories. For large imports, you may need to increase the `-mx32m` option to 64m (64 megabytes) if your machine has the required RAM.

Commit and exit

This is to commit all changes made since the last commit.

Exit without saving changes

This is to abort all changes made since the last commit.

Edit menu option

Undo

This function allows you to undo your changes up to virtually infinite times as long as you have not committed your changes.

Redo

This function allows you to redo a series of previously undone actions as long as you have not committed your changes.

Select all

When activated on the Diagram Editor, it will select all the objects drawn on the page. It is disabled on the Specification Editor.

Delete Graphics

This command deletes the object from the page, but its model object still exists in the database. Links from or to the deleted object will also be deleted from only the page. However, there are exceptions. The Bookmark, Image, Note and Shortcut objects have no model object behind them. They are regarded as graphics only. Therefore, a Delete Graphic command will remove them completely from the page and the database. You can still recover them by Undo-ing, but not after a commit is done.

Plug In menu option

Export/Import

The Export and Import facilities work together. They allow you to share your data among different developers or project teams. You can export graphics data as well as model data in your Elixir CASE repository into XML format, and then import them into another Elixir CASE repository.

The XML Export/Import comes complete with a DTD. The DTD is included in the distribution of Elixir CASE and should be placed in the directory holding the XML file so that the XML can be validated during import.

Note

Sun's XML parser must be on your classpath in order to import. The latest version is available at <http://www.javasoft.com>.

Figure 3.11 Export

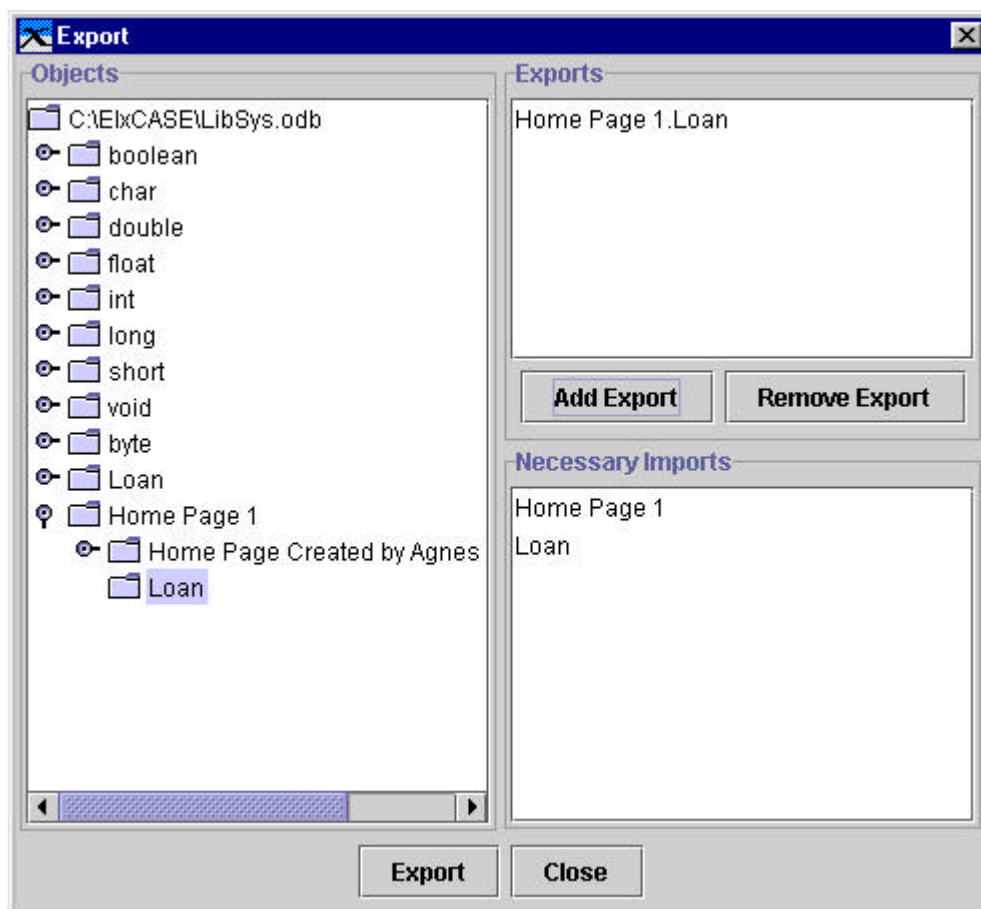


Figure 3.11 shows the Export dialog box. To export an object or diagram, select it from the list of Objects, and click on the “Add Export” button. The “Remove Export” button allows you to deselect your selection. In the example above, the *Loan* shape on the *HomePage* diagram is selected for export. (Note that this is not the *Loan* class).

The “Necessary Imports” box lists all the objects that you would also require when you import the selected object(s). In order to import the *Loan* shape into a database, *Home Page 1*, as well as the *Loan* class, has to be available in the database.

To export the objects, just click the “Export” button, and the “Close” button to close the Export dialog box.

Figure 3.12 Import

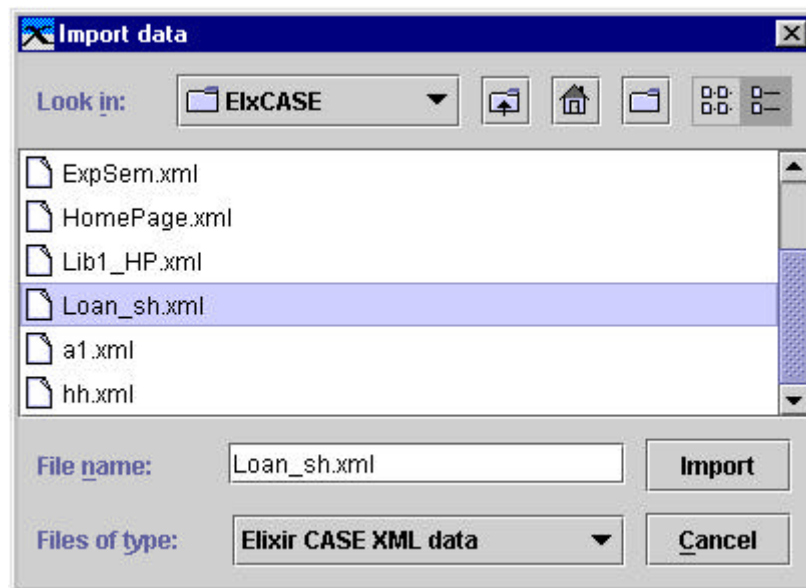


Figure 3.12 shows the Import dialog. An exported file has the extension `.xml`. In the diagram, the `Loan_sh.xml` is selected for import.

When an XML file is imported, it needs to locate the nodes to which the data to be imported is connected. In each XML file, there is an `<imports>` section which lists what needs to be found in the target database. Resolution is by node type (such as Class, Method, State, HomeDiagram, and so on) and name of the object.

In the case where the import function is not able to match the nodes, an Import Mapper dialog will appear to allow you to manually match the objects.

Figure 3.13 Import Mapper

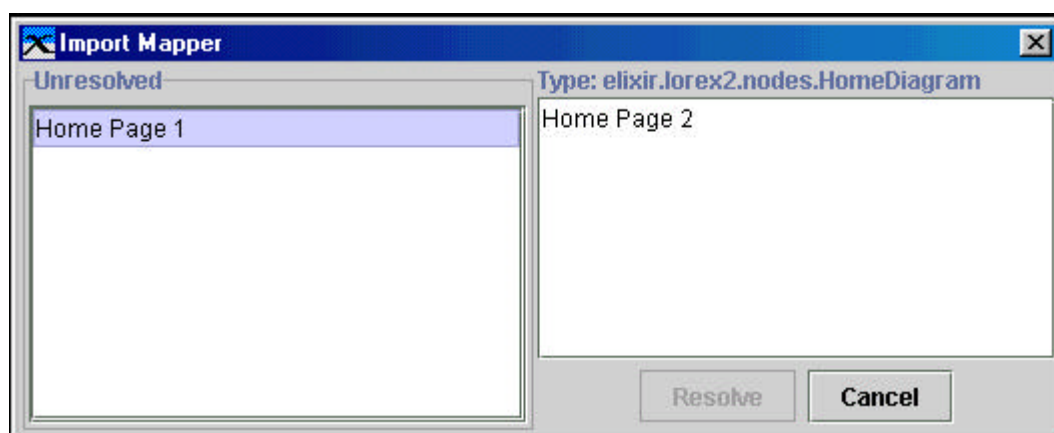


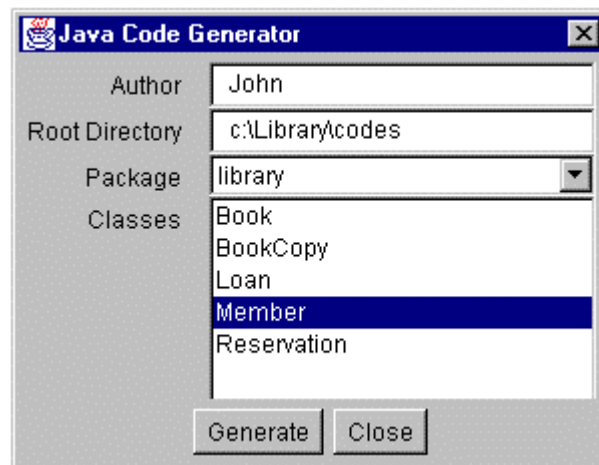
Figure 3.13 above shows the Import Mapper dialog. In the example above, *Home Page 1*, which is required for the import of *Loan*, is not available in the destination database. The available Diagram is *Home Page 2*. The Import Mapper provides you the option of matching *Home Page 1* to *Home Page 2*. Alternatively, you can abort the import by pressing the “Cancel” button on the dialog.

Java Code Generator

Java code generator is, as its name implies, to generate Java codes from the system that you have designed.

When this function is activated, the Java Code Generator window will pop up as in Figure 3.14.

Figure 3.14 Java Code Generator



By default, the Author field holds the name of the user logged into the system, but it can be changed.

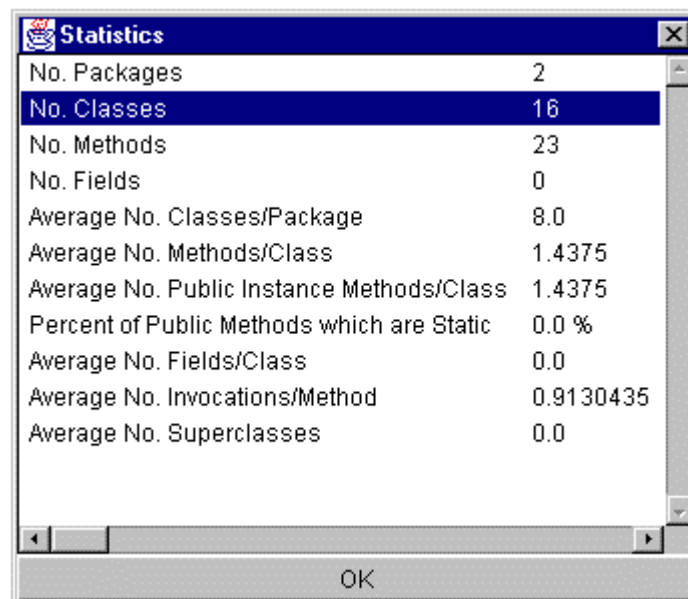
The root directory is where you can specify where the codes should be generated into, such as c:\Library\codes.

In the Package dropdown list box, you can see all the packages available in your system (includes imported packages as well as those you have created). Upon selection of one of the packages, the contents of the package will be displayed in the Classes area. Select by highlighting the class/classes for which you want to generate codes.

Now you can do a Generate. Upon completion, the codes will be in the directory that you have specified in “Root Directory”.

Statistics

Figure 3.15 shows the various software metric statistics that is generated in Elixir CASE. They can be compared against some industrial software metric standards to check if the designed system needs to be fine-tuned.

Figure 3.15 Statistics

No. Packages	2
No. Classes	16
No. Methods	23
No. Fields	0
Average No. Classes/Package	8.0
Average No. Methods/Class	1.4375
Average No. Public Instance Methods/Class	1.4375
Percent of Public Methods which are Static	0.0 %
Average No. Fields/Class	0.0
Average No. Invocations/Method	0.9130435
Average No. Superclasses	0.0

Go menu option

Back

... to previous page. It allows you to backtrack to all your previously visited pages.

Forward

... to the next page. It functions like the Internet web page forward function, where it allows you to move back and forth between pages.

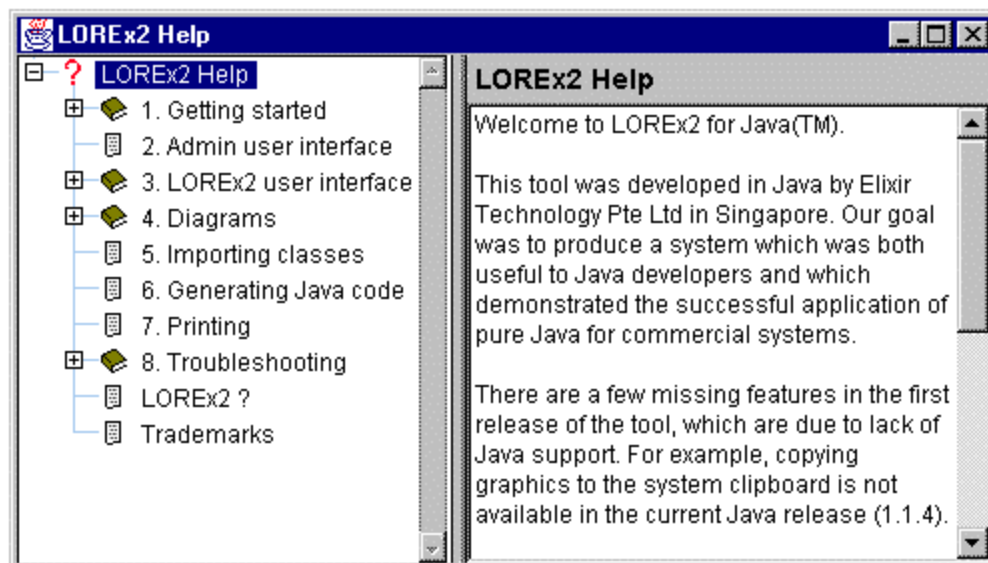
Home Page

This, of course, brings you back to the Home Page.

Help menu option

Help

This will call up the online Help, as shown in Figure 3.16. The online help is clearly structured, which makes for easy navigation. The topics covered are listed on the left, and the selected option will be described on the right.

Figure 3.16 Online Help**About**

This will provide general information on Elixir CASE tool, such as version number and the current logon user.

Great Finds in the Toolbox !

I bet you are eager to know what's in there ! If you're ready, let's dig in and get our hands dirty.

Figure 3.17 *Toolbox*

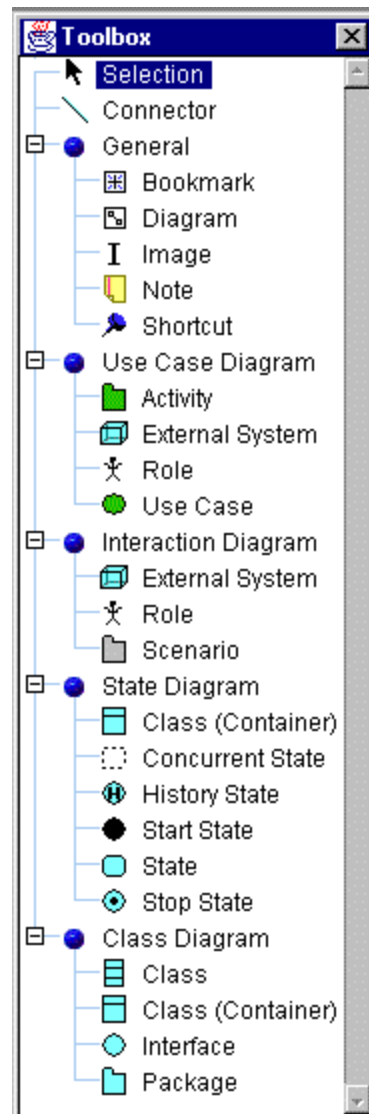


Figure 3.17 shows all the tools in the Toolbox. Basically, the Toolbox consists of all the *object types* that you can draw on your diagram page. The Toolbox is divided into 5 main categories, namely, General, Use Case Diagram, Sequence diagram, State Diagram and Class Diagram.

The tools under the General category are the only ones that need to be used across all diagrams.

The *object types* in the Toolbox, when drawn on the page, are referred to as *icons* for those under General category, and as *object graphics* for the other categories. The object graphics are the graphic representations of the *object models*, which holds the actual properties of the object stored in the database.

In this chapter, we will only describe the Selection and Connector tools and the tools in the General category; the tools in the other categories will be described in their respective chapters later.

Selection

The Selection tool, when highlighted, indicates that you are in the selection mode. In other words, you are now able to place your mouse on any object on the Diagram page and select it. We will normally be in this mode, unless one of the other tools in the Toolbox is selected.

Connector

The Connector tool in the Toolbox is used to link the various objects drawn on the various diagrams. It basically shows the relationship between the 2 objects that it is connecting. Depending on the objects being connected, the connector will take on different meanings and characteristics. For example, on a Statechart diagram, they are called *transitions*, while on an Object Sequence diagram, they are called messages. Only valid connections will be created between 2 objects. Therefore, the connector will be described as specific to its use in the subsequent chapters dedicated to each diagram.

General Objects

To place a general object on your diagram page, select the object from the Toolbox by using the mouse to left click on it once. It will be highlighted. Then position your mouse on any part of the page and left click once. Upon releasing the mouse, the object's icon will appear. Then you can use the green handles to resize your object as desired (applicable to Images and Notes only). Alternatively, You can click and drag an object to the required size. By right-clicking on the icon, you will see the options or commands available for that object. Here, we will only list all the commands that applies for each icon. A detailed explanation of the what and the how of the commands is given in Appendices A and B.

Bookmark

A Bookmark is used to mark a particular page or area on a page so that it can be easily accessed. It is to be placed on the top left-hand corner of the area where you want to 'bookmark'. It is recommended that you name the Bookmark such that you can easily know the contents of the bookmark area.

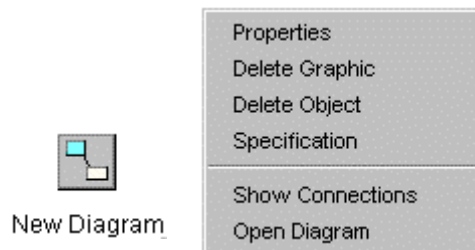
Available command that can be performed on a Bookmark :



Diagram

A Diagram is used when you want to create a new page. First, you need to place a Diagram object on an existing page. Then, when you double-click on the Diagram object, it will bring you to a new page. Similarly, it is recommended that you name the Diagram such that you can easily know what page it brings you to.

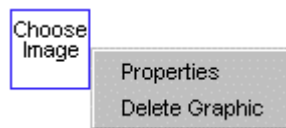
Available options or commands that can be performed on a Diagram :



Image

An Image is used when you want to display a picture on the page, for example, your company logo, product logo and so on. It accepts .gif and .jpg files.

Available options or commands that can be performed on an Image :



Note

You have seen a Note before on the Home Page. A Note is yellow in colour and is used for textual purposes.

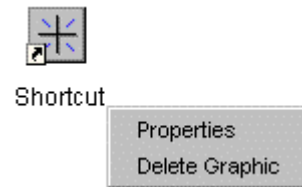
Available options or commands that can be performed on a Note :



Shortcut

This allows us to define a shortcut to a bookmark. You can have more than 1 shortcut to a bookmark for ease in navigation between diagrams.

Available options or commands that can be performed on a Shortcut :

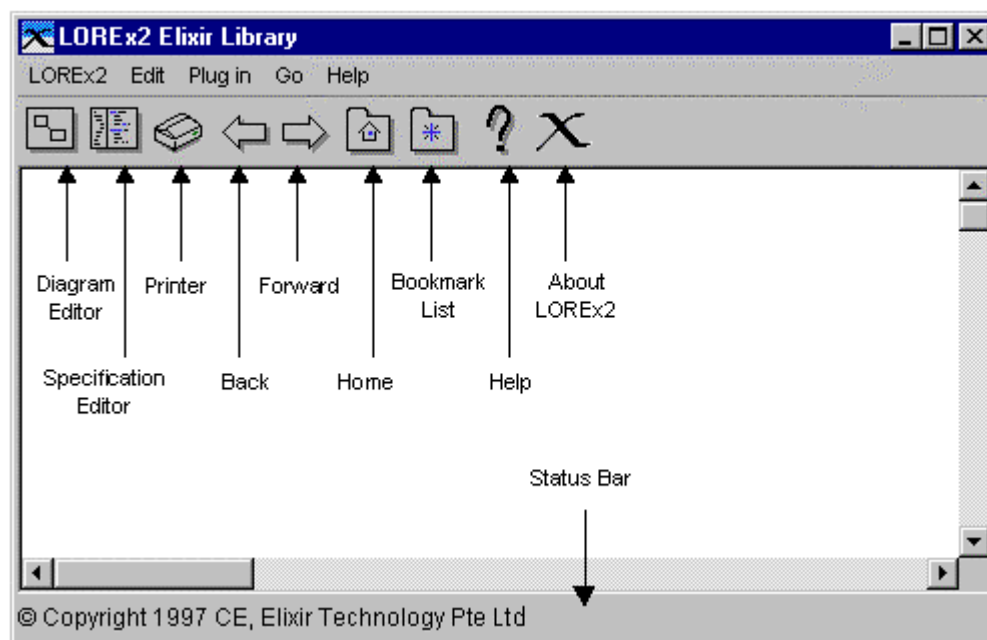


The Tool Bar is just a Click Away

The Tool Bar provides shortcuts to certain oft-used functions available under the Menu Bar and Toolbox.

Figure 3.18 shows all the tools in the Tool Bar with their function names.

Figure 3.18 The Tool Bar

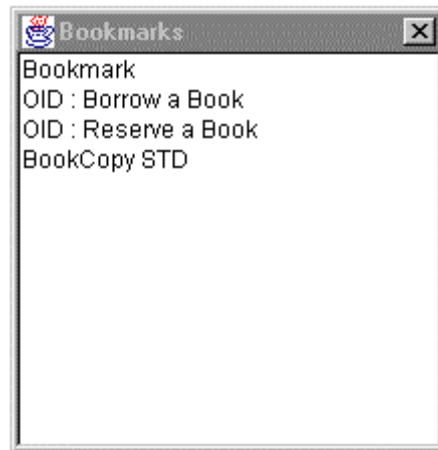


All the tools work the same way as described in the Menu Bar.

Bookmark Listing

Clicking on the bookmark object in the Tool Bar will bring up a Bookmarks window listing all the existing bookmarks for all the projects created in the database [See Figure 3.19]. One difference to note is that the Bookmark in the Toolbox is to create a bookmark, whereas the Bookmark List on the Tool Bar is to allow you to recall the bookmark(s), as described below.

Figure 3.19 List of Bookmarks



Selecting a bookmark will bring you to the area of the page where the bookmark object is.

What is the Status of ... ?

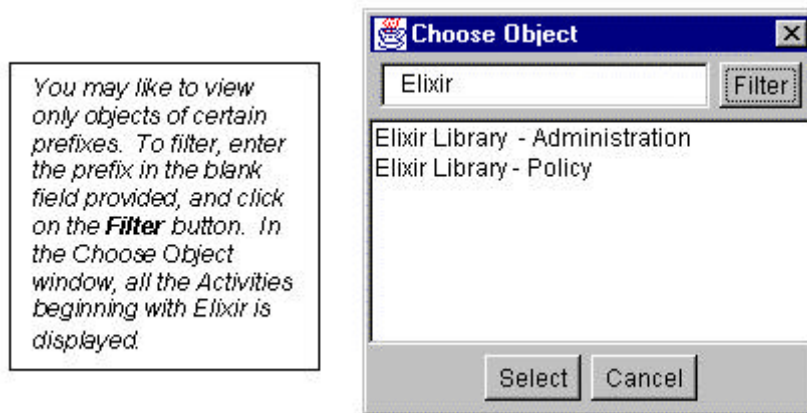
The status bar is at the bottom of the Elixir CASE window [See Figure 3.17]. It acts as a progress indicator for certain operations, such as importing, exporting and so on. It also reports any warning / error messages.

Across The Board

1. For all diagrams, if the container object is deleted, all the object graphics contained in it will also be deleted. However, the latter's object models still exists in the database. For the Use Case Diagram, the container object is the Activity; for the Object Sequence diagram, the Scenario; for the Statechart diagram, the Class (container); and for the Structure Diagram, the Package.
2. Elixir CASE does not impose any limit on the length of names of objects, be they classes, bookmarks, diagrams, connectors and so on.
3. Objects can be used on more than one diagram. Therefore, in order not to recreate the object, (which will give us 2 views of that object), we can do the following :
 - 3.1. Select the object type from the Toolbox;
 - 3.2. Hold down the <Ctrl> key and left-click the mouse on the page where you want the object to be;

- 3.3. The Choose Object window will appear, listing all the existing object instances of the selected object type (see Figure 3.20). Upon selecting one object instance, a view of the corresponding object graphic will then appear on your page.

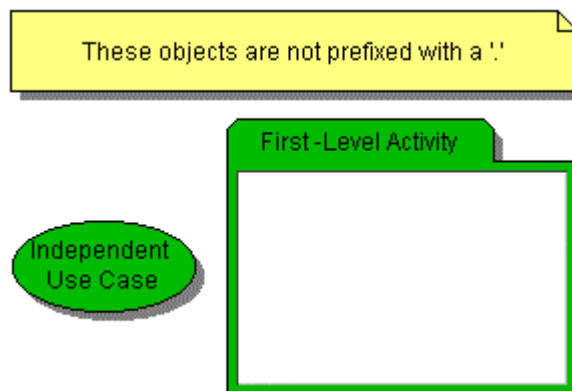
Figure 3.20 Choose Object Window



4. You may have noticed that, on the Toolbox, each time after you finished drawing a use case, say, on the page, the use case object type on the Toolbox will no longer be highlighted. Instead, "Selection" is highlighted. This is in expectation that you will not want to draw another use case immediately after. However, if you wish to do so, Elixir CASE provides the Shift key to overcome the default setting, as follows :
 - 4.1. Using the mouse, highlight the object type that you want to draw on the Toolbox.
 - 4.2. Then, pressing the Shift key, draw the object graphic on the page as per normal.
 - 4.3. When this is done, the object type on the Toolbox is still highlighted. You can continue to draw another of the same object type on the page.
5. If an object graphic on the page is selected (that is, you see green handles on it), you can actually use the arrow keys (← ↑ ↓ →) to move it around.
6. Naming Convention of Object Graphics on the Diagrams

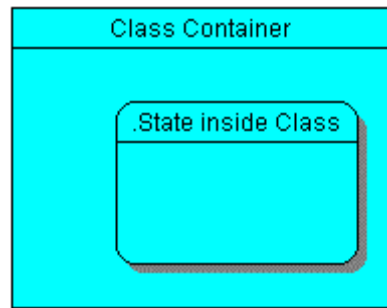
Objects on the diagrams are differentiated by the naming convention, as follows :

- 6.1. Independent Objects and First-level Container Objects



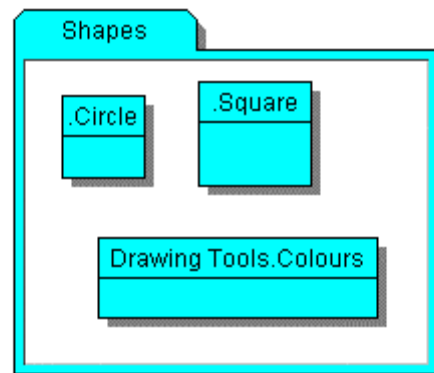
6.2. Objects in Containers

The State is prefixed with a '.'



6.3. Objects with a Parent but Drawn within other Containers

The class "Colours" belongs to the package "Drawing Tools" but is drawn in package "Shapes".



How to Quit from Elixir CASE ?

There are a few ways to quit from Elixir CASE, as follows :

1. You can choose to **Commit and exit** using the Menu Bar's Elixir CASE option.
2. You can also choose to **Exit without saving changes**, again using the Menu Bar's Elixir CASE option.

For Windows environment only

1. At the top left-hand corner, there is a Elixir CASE icon. Double click on it and you will be prompted as to whether you want to abort without saving the changes made. Choose as desired.
2. You can also click on the X sign at the top right hand corner of the window. You will then be prompted as to whether you want to abort without saving the changes made. Choose as desired.

What is Next ?

This chapter has explained to you all the tools and their usage. These tools are very flexible and it is up to you to fully make use of them in all ways that you can think of. They will facilitate your system development and make it very enjoyable.

Now that you are armed with all the necessary tools, you must be raring to go on, so what are we waiting for ?!

Chapter 4

Use Case Diagrams

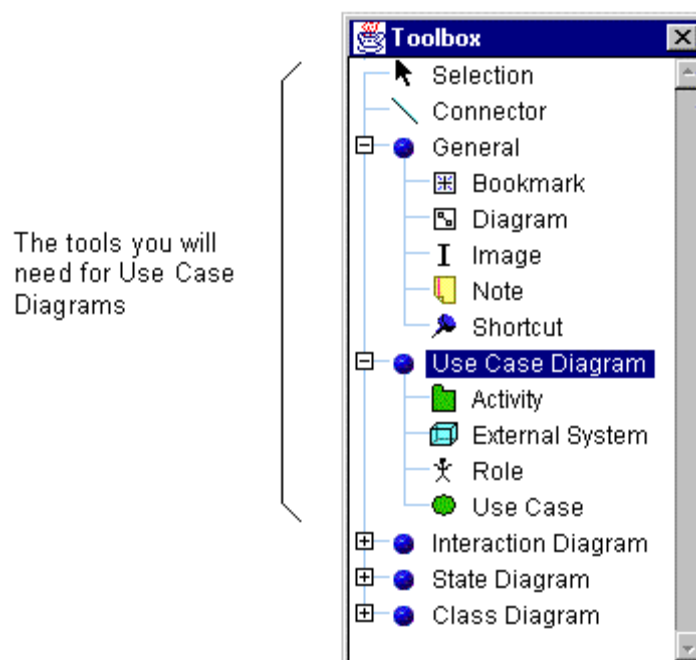
Use Case Diagrams are the first diagrams that we use in the analysis and design of a system. In this chapter, we will learn what is a use case diagram and what constitutes a use case diagram. We will also learn how to use the various use case diagram tools to assist us in our documentation.

In This Chapter

- ❑ Tools Used in Use Case Diagrams
- ❑ What is a Use Case Diagram ?
- ❑ Components of a Use Case Diagram
- ❑ How to Create a Use Case Diagram ?
- ❑ What is Allowed for Use Case Diagrams ?
- ❑ What is Next ?

Tools Used in Use Case Diagrams

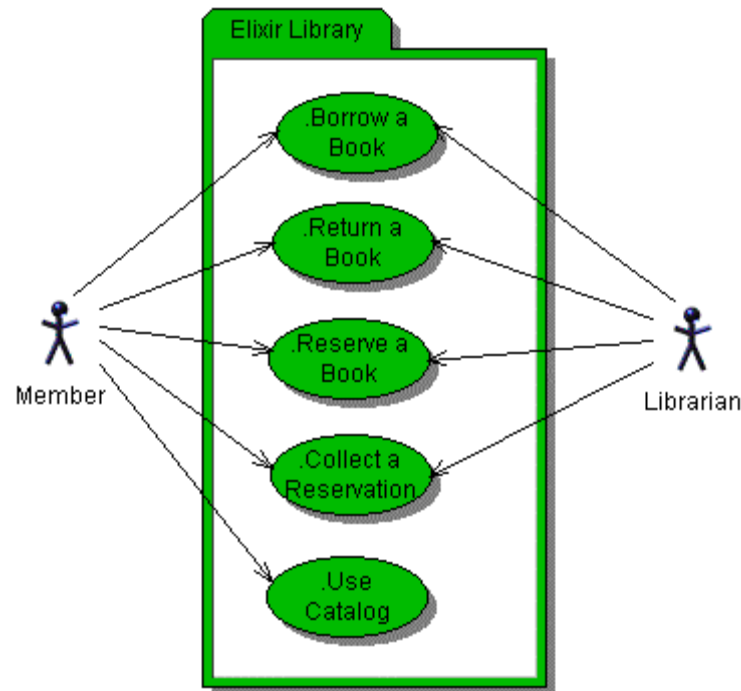
Figure 4.1 Tools for Use Case Diagrams



What is a Use Case Diagram ?

A Use Case Diagram is the top-level diagram of a system, providing an overview of the system requirements. It is a tool used to document all the functions (use cases) which are to be provided in the system. Figure 4.2 shows a Use Case Diagram of a Library System.

Figure 4.2 A Library Use Case Diagram



Components of A Use Case Diagram

A Use Case Diagram has the following components :

Role

A role represents the role of a person in the system. For example, in Figure 4.2, Member and Librarian are 2 roles in the Library System.

Use Case

A system needs to be broken down into discrete functions. Each of these functions will be represented as a use case. In Figure 4.2, "Borrow a Book", "Return a Book", "Reserve a Book", "Collect a Reservation" and "Use Catalog" are use cases.

External System

An external system is an object that interacts with the system but is not part of the system. It can be another system providing input to or accepting output from the system in question.

Activity boundary

An Activity boundary has 2 functions :

- To cluster the activities into logical groups;
- To separate the external systems from the use cases.

An Activity boundary is green in colour, as opposed to the other diagrams, which you will encounter in subsequent chapters.

How To Create A Use Case Diagram ?

Activity boundary

Firstly, we will need to draw the Activity boundary. Drawing the Activity boundary is like drawing a General object (Described in Chapter 3 – Great Finds in the Toolbox.General Objects). First, you have to select “Activity” from the Toolbox. Then, position your mouse on any part of the blank page and click your mouse once and drag it to the desired size. Upon releasing the mouse button, the Activity boundary will appear. Alternatively, you can simply just click and leave the resizing till later.

Use Case, Actor and External System

Using the same method, we can then add in the other objects. Use cases are drawn within the boundary while actors and external systems are drawn outside the boundary.

Note

The order and position in which you draw the objects is important. When you draw the activity boundary first before the use cases, with the use cases being within the boundary when first drawn, Elixir CASE will recognize that there is a parent-child relationship between the activity (parent) and the use case (child). When the relationship is correctly drawn, Elixir CASE will restrict the use case to just within the activity boundary. This parent-child relationship will be reflected in the Specification Editor.

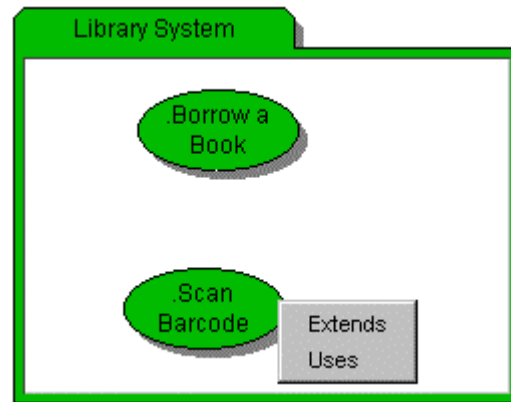
If a use case is drawn outside an Activity boundary, by default, the parent is the Diagram. Elixir CASE provides you with the flexibility of changing its parent to link it to an activity. This is done on the “Identity with Parent” tab page as described in Appendix B.

Relationship

A diagram is meaningless if we do not indicate the relationship between the various objects. On the Toolbox, there is an object called the “Connector”. This is used to connect the objects on the diagrams. Select it using your mouse. Then, position your mouse on the source object and drag it to the destination object. You can give a name to the relationship through the Properties window.

When you connect 2 use cases, a pop-up menu will appear to prompt you to name the types of connection. In Figure 4.3, upon connecting the 2 use cases “Borrow a Book” and “Scan Barcode”, the pop-up menu with “Extends” and “Uses” appears.

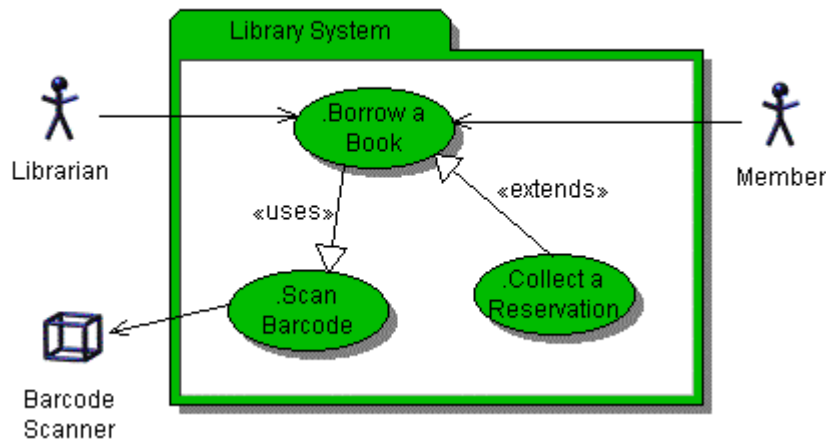
Figure 4.3 Pop-up Menu to Define Relationship Between Use Cases



There are 2 types of relationships, namely, “Uses” and “Extends”. In Figure 4.4, the use case “Borrow a Book” uses “Scan Bar Code”. Here “Scan Bar Code” is like a subroutine called by “Borrow a Book”. This can be very useful for extracting and sharing information.

Also, in Figure 4.4, “Collect a Reservation” extends “Borrow a Book”. This means that “Collect a Reservation” is a kind of “Borrow a Book”. The same basic sequence of behaviour defined by “Borrow a Book” is used, extended or modified by changes described in “Collect a Reservation”.

Figure 4.4 “Uses” and “Extends” Relationships



When an object that is connected is moved, the line will also move together with the object and the connection is still maintained. It works similarly for all other diagrams as well.

You can shape the arrow any way you like by adding nodes on the arrow. See Appendix A : Add Nodes and Delete Nodes.

Note

The objects painted on the screen are graphical representation of the object model in the database. Therefore, you can delete the graphical representation of the object using the “Delete Graphics” option in the popup menu (called up by right-clicking on the object). After “Delete Graphics”, the properties of the object you have defined are still stored in the database. You can paint the object on any diagram page again by selecting the corresponding object type from the Toolbox, hold down the <Ctrl> key, and then click on the page. This applies for all types of objects on all diagrams, not just for use case diagrams.

What is Allowed for Use Case Diagrams ?

1. More than one Use Case Diagram can be defined for a system. It can be used to break up the system into major groups of activities which are related. For example, an accounting system can be broken up into accounts payable, accounts receivable, general ledger and so on. In this way, it also facilitates the assignment of tasks for different groups of developers, with each group working on one or more Use Case Diagrams.
2. An activity can be embedded in another activity. This allows you to further categorize the use cases in an activity into logical groups.

What is Next ?

After completion of the Use Case Diagram, the next phase will be to expand on the use cases into Object Sequence diagrams. We will see how this can be done in the next chapter.

Chapter 5

Object Sequence Diagram

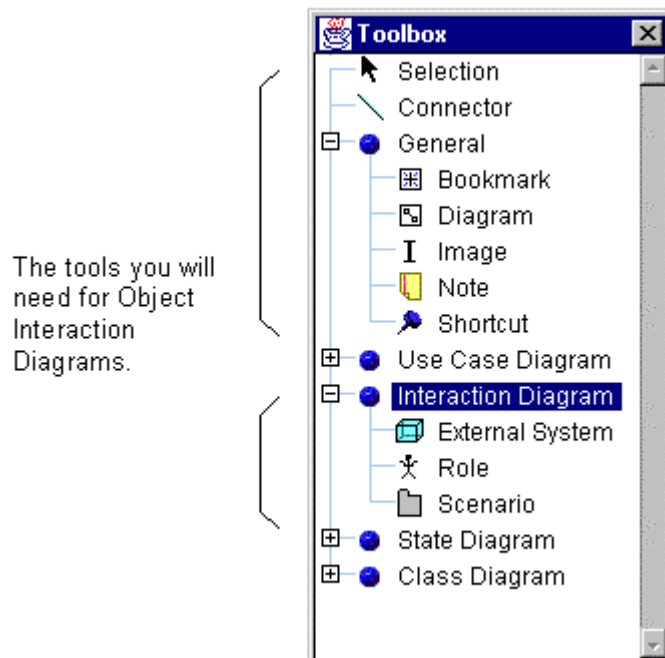
An Object Sequence diagram supports the modelling of object interactions and collaborations. In this chapter, we will learn how to create an object sequence diagram using the object sequence diagram tools in the Toolbox.

In This Chapter

- ❑ Tools Used in Object Sequence diagrams
- ❑ What is an Object Sequence diagram ?
- ❑ Components of an Object Sequence diagram
- ❑ How to Create an Object Sequence diagram ?
- ❑ What is Allowed for Object Sequence diagrams ?
- ❑ What is Next ?

Tools Used in Object Sequence diagrams

Figure 5.1 Tools for Object Sequence diagrams



What is an Object Sequence diagram ?

In the previous chapter, we learnt about use case diagrams. A use case is a component of a use case diagram. Each use case may comprise of one or more possible course of actions, which we call scenarios. For example, the use case “Borrow A Book” has the following scenarios :

- ◆ Borrow A Book - Successful
- ◆ Borrow A Book - Exceeds Loan Limit
- ◆ Borrow A Book - Outstanding Fine
- ◆ Borrow A Book - Exceeds Renewal Limit

Each scenario consists of a chronological sequence of actions. These actions are actually messages, the means by which objects involved in the scenario interact. All these actions for each scenario are represented on an Object Sequence diagram.

Components of An Object Sequence diagram

An Object Sequence diagram has the following components :

Role

A role represents the role of a person in the system. It is similar to the role in a use case diagram.

External System

An external system is an object that interacts with the system but is not part of the system. It can be another system providing input to or accepting output from the system in question. It is also similar to the external system in a use case diagram.

Scenario boundary

A Scenario boundary has 2 functions :

- To separate the system into discrete functions for analysis
- To separate the external systems from the classes.

A Scenario boundary is grey in colour.

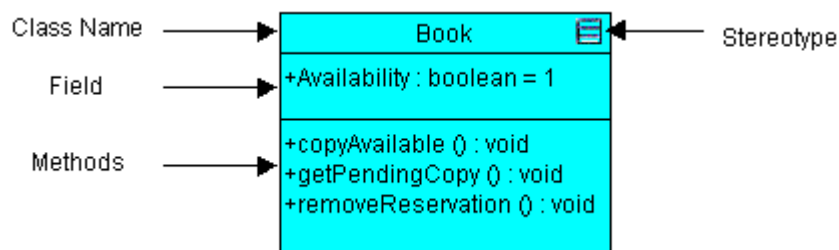
Class

A class is a structure that contains fields, methods and semantics. It is the basic building block of an object-oriented system.

The class symbol is a blue rectangular box. The box can be segmented up to 3 sections, one above the other. The sections are described as follows :

Section	Description
Class Name	Name used to identify the class and is thus compulsory.
Fields	The characteristics belonging to the class.
Methods	The methods available for the particular class. Figure 5.2 shows some examples of fields and methods that are provided by the class “Book”.

Figure 5.2 Fields and Methods available in Class “Book”



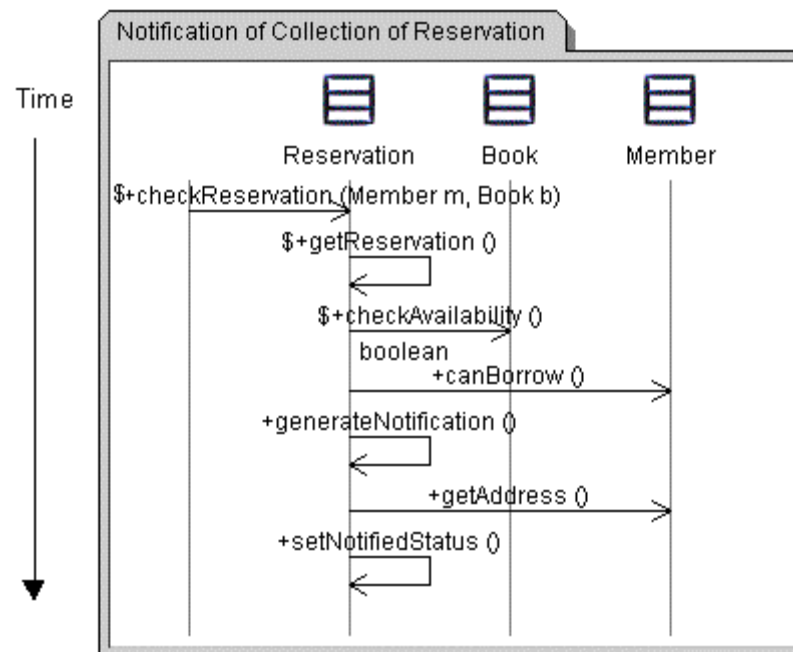
A class may be classified. We do this using class Stereotypes. Stereotypes are represented at the top right-hand corner of a class object, as shown in Figure 5.2. Stereotypes are useful in that it shows at a glance what kind of function the class performs. The stereotype may also be incorporated into the codes during code generation.

By default, the stereotype and the fields and methods of the class are not shown on the class object. You can choose to do so by using the Show tab page. Please refer to Appendix B for a detailed description of its usage.

Lifeline

A lifeline is the line that runs vertically under an object. It represents a time line, moving from top to bottom, as shown in Figure 5.3.

Figure 5.3 Scenario to Notify Member of Availability of Reserved Book



Message

A message is represented by an arrow flowing from one lifeline to another. It is the means by which an object invokes / activates another object. It is labelled with the name of the service to be requested, which we will call `MessageName`. Figure 5.3 shows the use of the various message notation.

Message Notation :

Notation	Syntax	Description
Argument Passing	<code>MessageName([arg 1,...,arg n])</code>	To pass arguments between the objects.
\$	<code>[\$]MessageName</code>	Indicates that the message is sent to a class rather than an instance of the class.
Return Values		Returned messages are implicit and are not represented by arrows. The return type of a method is indicated at the arrow, as shown in Figure 5.3, where <code>Book.checkAvailability</code> returns a <code>boolean</code> value.

A message to self

An object can send messages to itself. An example of when an object would need to call itself is when it requires another service that itself provides. In Figure 5.3, the Reservation class calls itself 3 times, firstly, to *getReservation*, and then to *generateNotification* and lastly, to *setNotifiedStatus*.

How To Create An Object Sequence diagram ?

Scenario boundary

First, you will need to draw the scenario boundary by selecting from the Toolbox. On the scenario boundary, there will be a vertical line on the left side. This is the initial lifeline, from which the first method is triggered.

Create Classes

You may be wondering how you can draw the classes. To do that, you will first have to create the class models in the system. I suggest that you create the classes on the same diagram but draw them outside the Scenario boundary. In this way, you can easily access them to, say, add methods or fields. Then, after completing the scenario, you can use "Delete graphic" to delete the object graphic from the diagram page.

Note

As I have mentioned in Chapter 3, the class objects that are drawn on the Diagram page are just graphical representation of the class models in the database. Through these graphical class objects on the page, we are able to define the various properties of the class model and these properties will be stored in the database. If we delete these graphical objects from the page, the class models still exist in the database. We can draw another graphical class object on some other pages to access the class model definitions (using Ctrl-click).

To create a class, select the Class tool which is under "Class Diagram" or the Class (container), and draw it on the page as per normal. Give the class a name. Go into the properties for the class to define the methods, fields and other characteristics. These characteristics are described in Appendix B, under the various Property tab pages.

Note

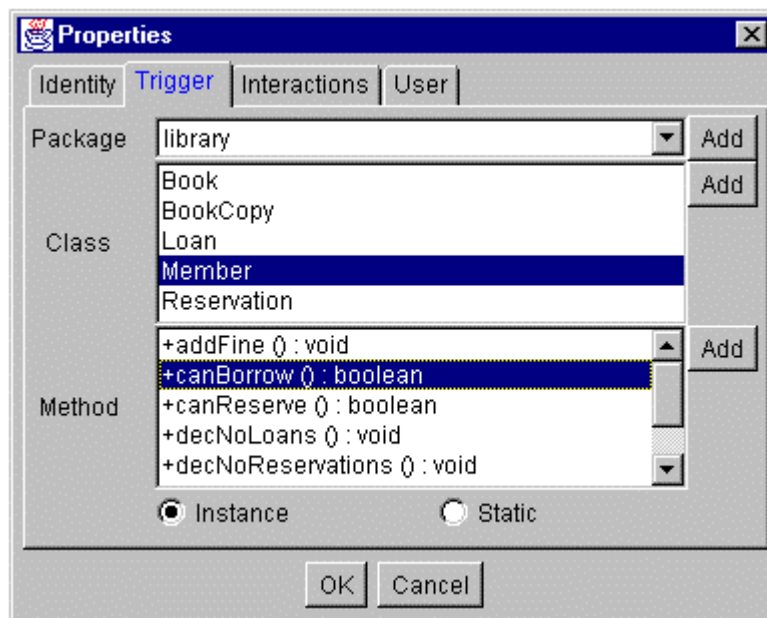
At this stage, you must at least define the methods provided by the classes to be used in the object sequence diagrams.

Draw Classes

Now, we are ready to draw the classes and the messages that are passed between classes. We will illustrate using the “Borrow a Book” Object Sequence diagram of the Elixir Library System. The classes involved are Member, Loan and BookCopy. They have already been created.

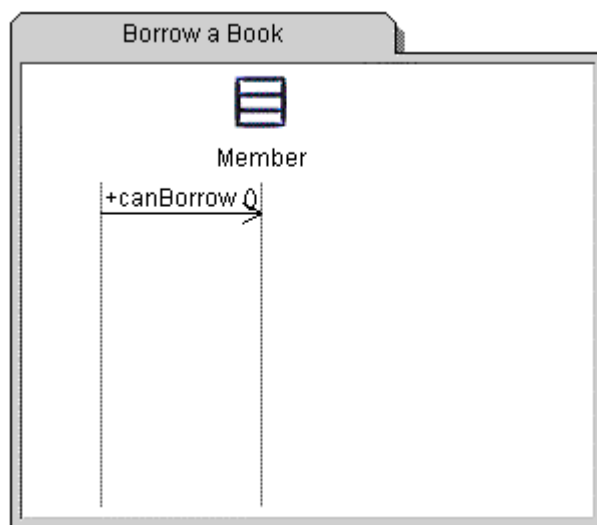
First, we need to specify the entry function into a Scenario method, that is, the method that triggers the “Borrow a Book” scenario. This is done on the Trigger tab page under the Properties window of the Scenario boundary, as shown on Figure 5.4 [Hint : Right-click or double-click on the Scenario object to get the Properties window]. The various features of the Trigger tab page are described in detail in Appendix B.

Figure 5.4 Trigger tab page



Choose the *Library* package and you will see all the classes in this package. Select the *Member* class and its methods will be listed in the Method area. Select the *CanBorrow()* method and the Instance radio button, and press OK. The result is shown in Figure 5.5.

Figure 5.5 Partial “Borrow a Book” OID



Now, we need to define the interaction of the *Member* class with other classes. Go into Properties again, this time choosing the Interaction tab page [Refer to Appendix B for a detailed description of the usage of the Interaction tab page]. The Sender field shows the class and its method that is calling for or invoking the service of the next class. The Add button allows us to add an invocation. Upon pressing Add button, the Add Invocation window appears, as shown in Figure 5.6. From there, we can choose the method of the required class in a particular package for invocation. In our example, the class *Member* invokes the *Create* method of the class *Loan*. The resulting diagram is shown in Figure 5.7.

Figure 5.6 To add further invocations

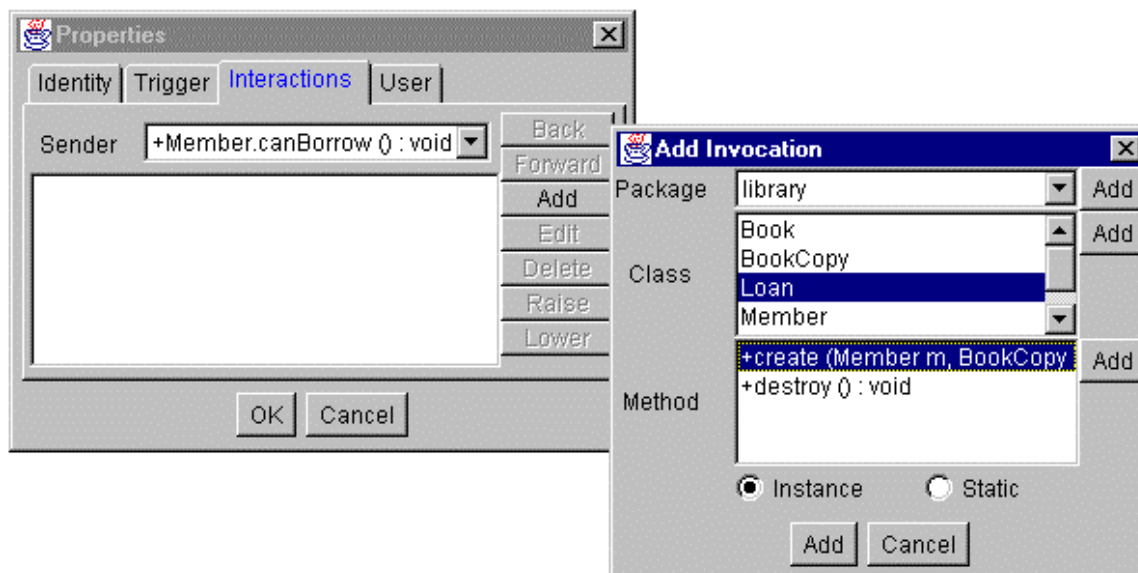
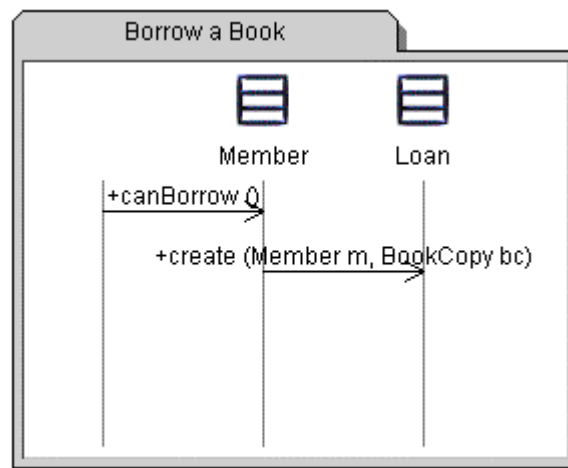


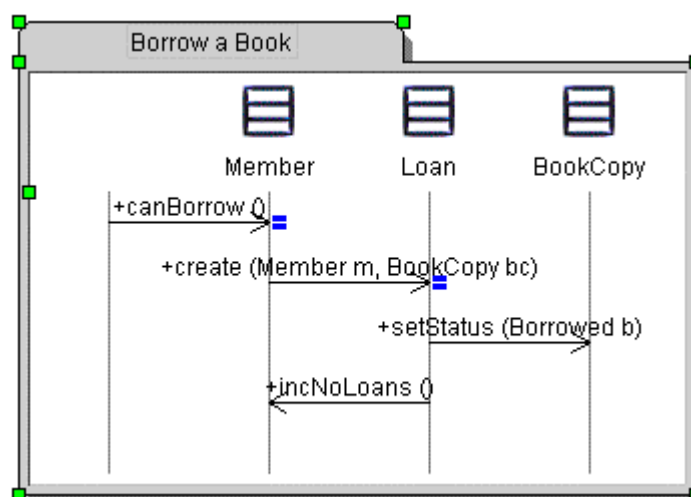
Figure 5.7 *OID after the invocation of Loan:create(Member, BookCopy)*



Subsequent invocations are also done using the Interactions tab page, including messages to self.

The final Object Sequence diagram will be as in Figure 5.8. Notice that at the end of the CanBorrow() message line and the Create() message line, there is each a small blue box with a '-' sign. These blue boxes will only appear when the Scenario object is selected. If you click on it, the rest of the classes and messages sent from this message will be hidden. Click on it again, and they will be displayed again. This is useful especially on an Object Sequence diagram that is complicated; it can be used to hide lower level interactions in order to focus on specific classes and methods on the diagram.

Figure 5.8 *Completed "Borrow a Book" OID*



Note

It is possible to import Java classes to be used in your diagrams. Chapter 3 (Import Classes under Elixir CASE menu option) covers how to import classes.

What is Allowed on An Object Sequence diagram ?

A message can be an instance or a static. This is defined on Methods.General tab page of a class [refer to Appendix B]. You can then invoke an instance or a static message of a class on an object sequence diagram. This can be done on the Interactions.AddInvocation tab page of the Properties window of the class by choosing either the Instance or Static radio button [See Figure 5.6].

What is Next ?

At this stage, we should have documented most of the processes of the system. The next phase will be to pick out the major classes which is involved in many of the processes and make an analysis of the different states it goes through in its life span. These will be modelled using a Statechart diagram. We will cover this next.

CHAPTER 6

Statechart Diagram

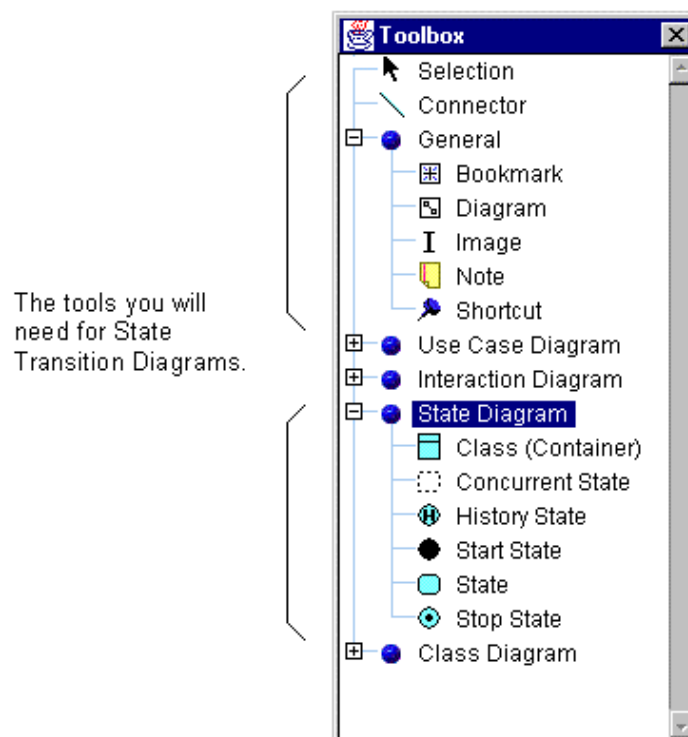
A Statechart diagram models the different states of objects in a system. In this chapter, we will learn how to create Statechart diagrams using the Statechart diagram tools in the Toolbox.

In This Chapter

- ❑ Tools Used in Statechart diagrams
- ❑ What is a Statechart diagram ?
- ❑ Components of a Statechart diagram
- ❑ How to Create a Statechart diagram ?
- ❑ What is Allowed for Statechart diagrams ?
- ❑ How to Use Start State, Stop State and History State ?
- ❑ Simplification Techniques in Textual Specification
- ❑ What is Next ?

Tools Used in Statechart diagrams

Figure 6.1 Tools for Statechart diagrams



What is a Statechart diagram ?

Object Sequence diagrams model the behaviour of the system, and in the process identifies all the objects in the system. From an object sequence diagram, we can see the flow of information from one object to another. This flow of information constitutes a process. An Object Sequence diagram therefore, provides an overview of the interaction among the objects in the system.

In contrast, in a Statechart diagram, the focus is on each individual class in the system. Imagine you're a doctor dissecting the body (that is, the system) and picking out each major part of the body (that is, the classes) to further examine it. Likewise, a Statechart diagram focuses on one class and models the different states that the object goes through and the events that trigger its change in states. We can identify the different states of an object from studying the object sequence diagrams, such as the pre-conditions and post-conditions for each scenario.

At the same time, Statechart diagrams also complement Object Sequence diagrams in that they verify the completeness of the latter. For example, upon further analysis of a class, we may discover a possible event that has been overlooked in the object sequence diagrams.

Components of A Statechart diagram

A Statechart diagram (STD) basically consists of states and transitions. There are some special states which are represented by specific notations for clarity and simplicity. The components are described as follows :

Class (Container)

A class container is just an alternative graphical representation of a class. On a statechart diagram, a class container is used to "contain" states. A Class (Container) has 2 distinct sections :-

Section	Description
Name	To identify the class.
Body	This is the main part of the class (container) where the different state transitions of the class is modelled.

State

A state has 3 distinct sections :-

Section	Description
Name	To identify the state box.
Body	Sub-states are modelled in this section.
Textual	This section is where we can define formal specifications. This is a form of textual documentation used to further clarify or to simplify the state diagram. Elixir CASE provides 3 clauses to simplify the statechart diagram, namely, the entry/, exit/ and do/ clauses. They will be covered later in this chapter.

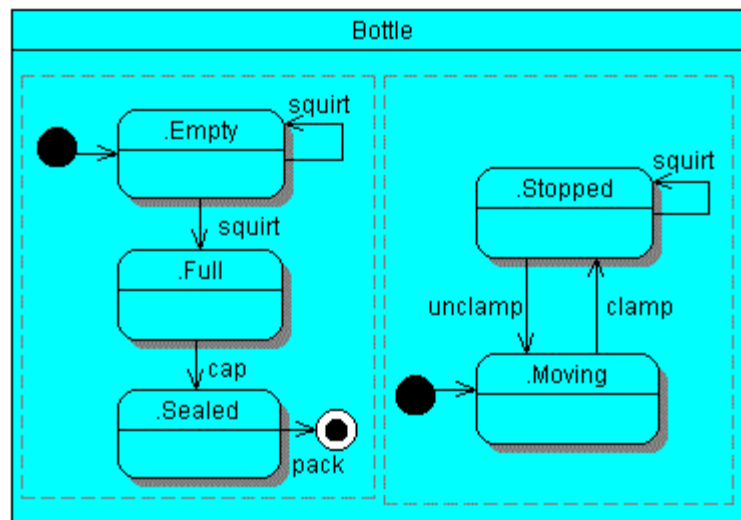
The Body and the Textual sections can be hidden or shown by pulling down the green handle in the middle of the state object.

Usually, we use an adjective to name a state.

Concurrent State

At one point of time, a class may be involved in different kinds of activities, leading to the possibility of it being in different kinds of states, which may or may not be related. For example, in Figure 6.2, the Bottle class is concurrently in 2 states. The bottle will continue moving and stopping through the production line, regardless of whether it is empty, full or sealed, until it is packed and reaches its stop state.

Figure 6.2 Concurrent States



In a STD, there can be any number of concurrent states, but a realistic limit is probably 2 or 3. Each concurrent state can have its own independent start and stop states.

Start State

A start state is used to indicate the original state of the class. Every STD should have a start state.

Stop State

A stop state is used to indicate the final state of the class. An STD may not always have a stop state object as there are various ways to indicate stop states. Refer to the section below on "How to Use Start State, Stop State and History State ?"

History State

The history state stores the state of the class just before exiting from that state. This is so that upon re-entry, the system can remember what was the last state at which it left and be able to resume from there.

Transition

A transition is the action or event that causes the class to change its state. They are actually the various scenarios that we have created in the use case diagrams.

Usually, we use a verb to name a transition. This also helps to differentiate between a state and a transition.

How To Create A Statechart diagram ?

Class Container

Firstly, we will need to draw the class container [Hint : Use <Ctrl>- click to verify if the class already exists].

State Objects

Next will come the various state objects. There is no drawing order among the various state objects. However, they must all be drawn within the class container. In fact, Elixir CASE prevents them from being drawn outside.

The state graphic will originally show only 2 sections, the Name section and Body section. If you look closely at the bottom centre, there is a green handle. Use the mouse to push the green handle up and the textual section will be revealed.

Note

The order and position in which you draw the objects matter. You have to draw the class (container) first before the states. Also, the states have to be within the class (container) when first drawn. When you do this, Elixir CASE will recognize that there is a parent-child relationship between the class (container) and the state. In fact, Elixir CASE restricts the state to just within the class (container). This parent-child relationship will be reflected in the Specification Editor.

Transitions

The transitions, like all links, can be drawn using the Connector. Refer to Transitions tab page in Appendix B for a detailed description on defining the characteristics of a transition.

You can shape the arrow any way you like by adding nodes on the arrow. See Appendix A : Add Nodes and Delete Nodes.

What is Allowed on A Statechart diagram ?

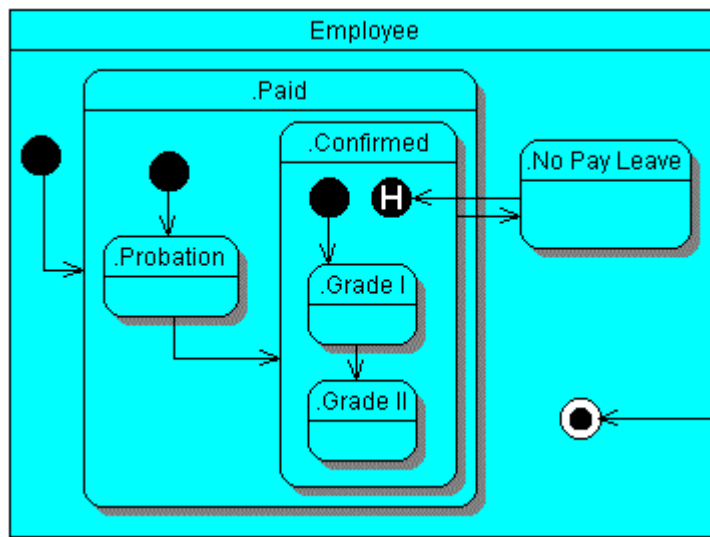
1. It is possible to have sub-states or nested states in a statechart diagram. A nested state is no different from any other states; each nested state can have its own start state, stop state and history states. It can even have other states nested within itself.
2. It is possible for a state to transit back to itself. In Figure 6.2, the *Empty* state transits back to itself by the *squirt* action.
3. It is possible to have transitions of the same name originating from different sources. In Figure 6.2, the *squirt* transition originates from both the *Empty* and *Stopped* states.

How to Use Start State, Stop State and History State ?

Start State

A start state will always point to the original state. As each nested state can have its own start state, it is possible to have a STD as in Figure 6.3.

Figure 6.3 *Start States*



History State

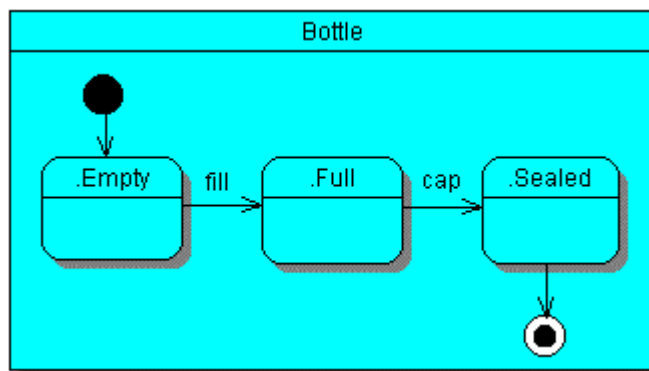
A history state is drawn within the state box that has 2 or more exit values. There will be a re-entry arrow from an external state box pointing to the History state. In Figure 6.3, the 2 possible exit states are Grade I and Grade II. By using a History state, we can remember at what state we left and will be able to resume from there upon re-entry.

Stop State

Stop state means that the class has reached the end of its life span. There are 3 ways to represent a stop state :

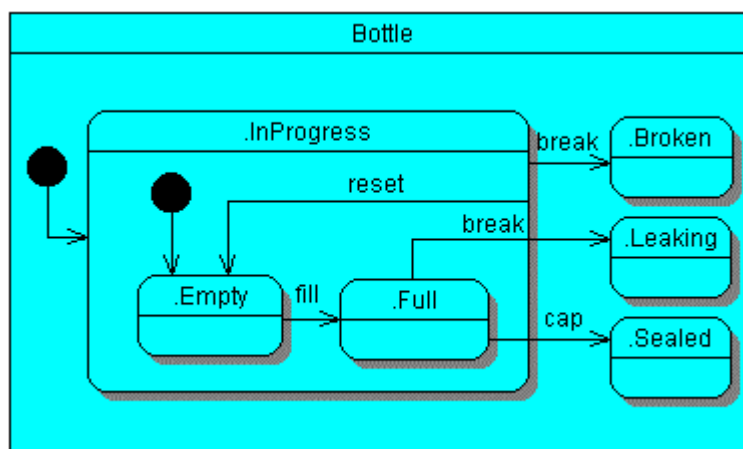
- i) A simple case is where a class has only one distinct stop state, as shown in Figure 6.4, where *Sealed* is the stop state.

Figure 6.4 One Distinct Stop State



- ii) It is possible that a class reaches the end of its life span at different states. In this case, we need not use the Stop State symbol as it may clutter up the whole state diagram. A state with no arrows coming out of it also represents a stop state. In Figure 6.5, Broken, Leaking and Sealed are the 3 possible stop states.

Figure 6.5 State Diagram with Implicit Stop States



- iii) A class can enter into a stop state regardless of the sub-state it is in. This is best illustrated in Figure 6.3. An employee can cease to be an employee whether he is a confirmed staff or still on probation, or whether he is paid or on no pay leave. This is represented using an incoming arrow originating from the *Employee* class (container) to a stop state.

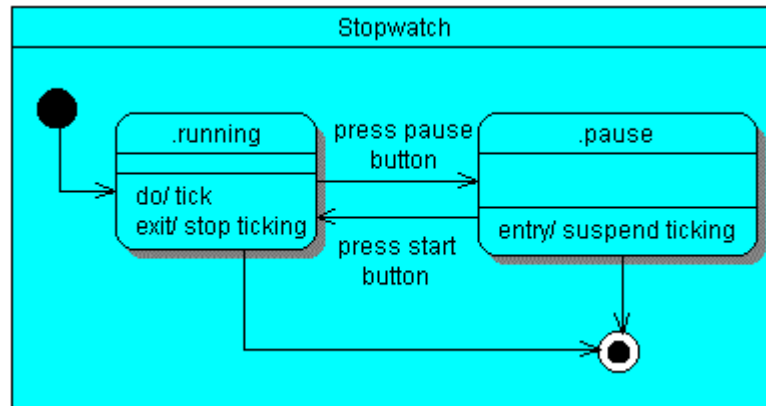
Simplification Techniques in Textual Specification

Rather than documenting repetitive events in the body section, Elixir CASE provides a mechanism to document them in the textual section. They come in the form of clauses, namely, entry/, exit/, and do/. They are explained in the table below :

Clause	Description
Entry	Allows the triggering of an event on all entries into a state. Syntax : entry/ <i>EventName</i>
Do	Allows an action to be performed continuously while in a state. Syntax : do/ <i>EventName</i>
Exit	Allows the triggering of an event on all exits out of a state. Syntax : exit/ <i>EventName</i>

The use of simplification clauses greatly reduces the clutter in the body section. Figure 6.6 illustrates how clauses are used.

Figure 6.6 The State Transition of Class “Stopwatch” using clauses



What is Next ?

We have now gathered most of the information about the system. Using the information, we should be able to derive a class diagram, which is what we are going to cover next.

Chapter 7

Class Diagram

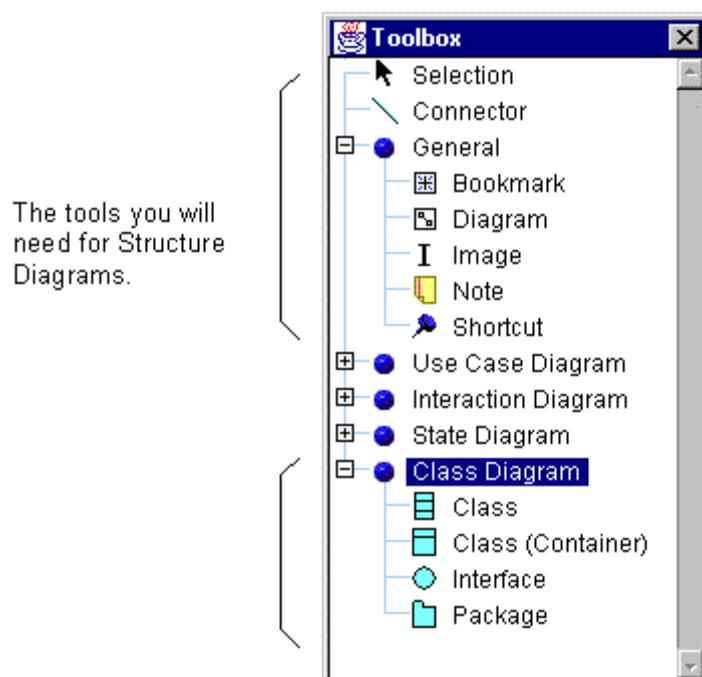
A Class Diagram allows us to capture the static relationships between objects. This includes association, inheritance and dependency relationships. This chapter explains the various components of a Class Diagram and how to use the Class Diagram tools to create a Class Diagram.

In This Chapter

- ☐ Tools Used in Class Diagrams
- ☐ What is a Class Diagram ?
- ☐ Components of a Class Diagram
- ☐ How to Create a Class Diagram ?
- ☐ What is Allowed for Class Diagrams ?
- ☐ What is Next ?

Tools Used in Class Diagrams

Figure 7.1 Tools for Class Diagrams



What is a Class Diagram?

In object-oriented concept, an object contains the name, type, attributes and services of the object. It also includes the relationship of that object with other objects in the system. Previously, we have seen in the object sequence diagram how one object calls for the services of another object through message passing. From an analysis of all the object sequence diagrams, we are able to transform these relationships onto first-hand Class Diagrams. The statechart diagrams will come into play to refine the Class Diagrams. In short, a Class Diagram is a detailed combination of the object sequence diagrams and statechart diagrams, giving a complete picture of how the different objects in a system relate to each other to form a complete workable solution.

Components of A Class Diagram

A Class Diagram has the following components :

Class

A class is a structure that contains fields, methods and semantics. It is the basic building block of an object-oriented system.

The class symbol is a blue rectangular box. The box can be segmented up to 3 sections, one above another. The sections are described as follows :

Sections	Description
Class Name	Name used to identify the class and is thus compulsory.
Fields	The fields and data belonging to the class.
Methods	The methods available for the particular class. Figure 5.2 in Chapter 5 shows some examples of fields and methods that are provided by the class "Book".

A class may be classified. We do this using class Stereotypes. Stereotypes are represented at the top right-hand corner of a class object, as shown in Figure 5.2 in Chapter 5. Stereotypes are useful in that it shows at a glance what kind of function the class performs. The stereotype may also be incorporated into the codes during code generation.

Class (Container)

A class container is just an alternative graphical representation of a class. On a class diagram, a class container is used to "contain" other classes; that is, you can create a class/classes inside a class container. For example, the class "Dialog" may contain an inner class "ButtonPressBehaviour". As we have seen in the previous chapter, a class container can also contain states in it.

A Class (Container) has 2 distinct sections :-

Section	Description
Class Name	To identify the class.
Body	To contain the inner classes, if any.

The difference between a class and a class container is that a class shows all the fields and methods of a class while a class container is able to show the inner classes and states of the class.

Interfaces

An interface is a class with no implementation, and therefore, has method declarations but no method bodies and no fields.

Package

A package is a mechanism used to break down a system into smaller, more manageable subsystems. Classes, which are associated to each other in such a way that they model a logical process or module of the system, are grouped into packages.

A dependency between 2 packages exists if any dependencies exists between any 2 classes in the packages.

2 or more packages may be interdependent on each other and they exist on the same level. Because of this interdependency, it is possible to have cyclic dependencies, which is also supported by Elixir CASE.

Relationship

The relationship between 2 objects is represented by a line linking one object to another using the Connector tool from the Toolbox.

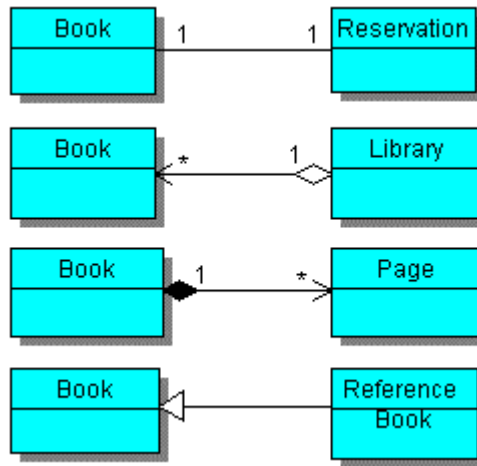
There are 2 basic types of relationships between classes, as follow :

Association

The relationship between 2 objects is referred to as an Association. For example, a book has a reservation; a person works for a company; a company has a number of offices.

There are 2 specific kinds of association relationships, namely, Aggregation and Containment relationships. Figure 7.2 shows the different types of association.

Figure 7.2 Types of Associations on a Class Diagram



An **Aggregation** is a special form of association that specifies a part-of relationship. It implies a constraint on the life-span of the component part. For example, a book is part of a library.

A **Containment** relationship is a stronger variation of the aggregation relationship. In a containment relationship, the part object is meaningless if it does not belong to only one whole object. In addition, deletion of the whole object means that the part is also deleted with it. A book and a page have a containment relationship; a book contains pages and destruction of the book destroys the pages as well. Also, the book knows its pages, indicated by the arrow from book to page, as shown in Figure 7.2.

Extends

An **Extends** relationship is equivalent to the inheritance relationship. It is a taxonomy relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element and contains additional information. An instance of the more specific element may be used where the more general element is allowed. For example, a reference book is a kind of book, as shown in Figure 7.2.

How To Create A Class Diagram ?

Package

First, you may like to provide a package to contain the classes. We can do that using the Package tool from the Toolbox to create a new package or to call up an existing one.

Class Container and Class

Next, we can add in the classes or class container as required. Most classes might have been created before, therefore, do check before creating a new class as we do not want to have 2 definitions of the same class [Hint : Do a <Ctrl>-Click to create a graphical view of an existing class].

A class container is a class. Therefore, if both class container and class object graphics are required for the diagrams, creation of the class needs to be done only once, either through the class tool or class container tool on the Toolbox. Therefore, do be careful when creating objects; not only for classes but other objects as well.

Note

The order and position in which the objects are drawn makes a difference, graphically as well as internally in the database. When you draw the package first before the class with the class being within the package, Elixir CASE recognizes that there is a parent-child relationship between the package and the class. Elixir CASE will also restrict the class to be within the boundary of the package. This parent-child relationship will be reflected in the Specification Editor.

However, it is not always possible to decide at the start this parent-child relationship between objects. Thus, Elixir CASE provides a function to support re-parenting of objects. A class created without a specific parent package will be first put into a default package. Later, you can re-parent it into the appropriate package. To do so, refer to the "General" tab page of a class in Appendix B.

Relationship between classes

When you connect 2 classes, a pop-up menu will appear to prompt you to name the connection. In Figure 7.3, upon connecting the 2 classes "Book" and "Regular Loan Book", the pop-up menu with "Extends" and "Association" appears.

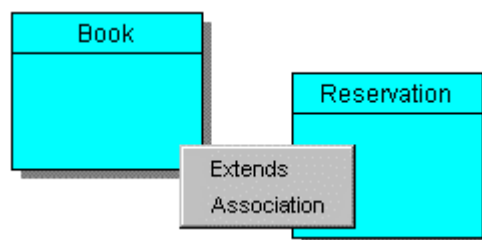
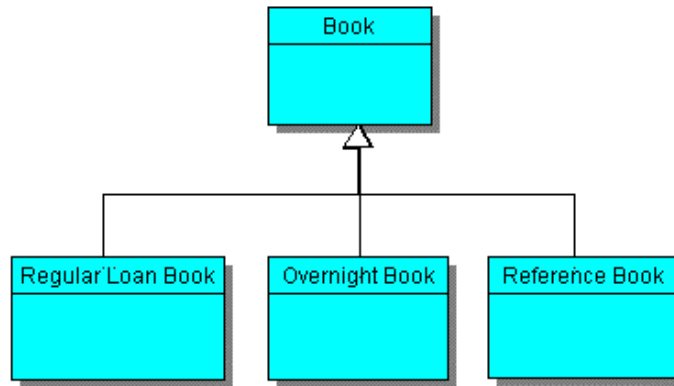


Figure 7.3 Pop-up Menu of a connector linking class to class

Extends

As explained before, when a class extends another, it inherits the properties of the class it extends. In Figure 7.4, the classes “Regular Loan Book”, “Overnight Book” and “Reference Book” extends class “Book”.

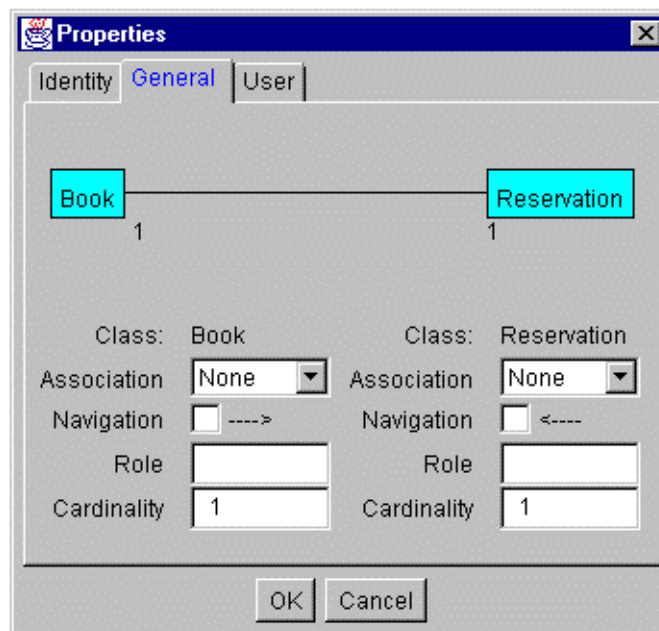
Figure 7.4 Extends relationship



Association

Upon choosing *Association* from the popup menu, the Properties window of the connector appears on the screen. On the General tab page, we can further specify the relationship between the 2 classes. The tab page shows the 2 classes, one on the left and the other on the right. For each class, we can specify the type of association, specify the directional knowledge (navigation), name the role and specify its cardinality with respect to the other class. As you specify the association, they will be shown graphically on the tab page, as in Figure 7.5.

Figure 7.5 Association between Classes “Reservation” and “BookCopy”



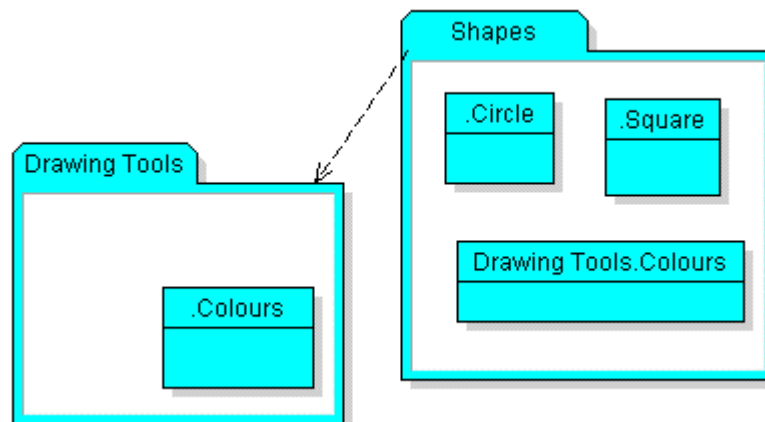
Relationship between packages

In a class diagram, there may be more than one package. If there are dependencies among the classes in the packages, we may like to explicitly indicate on the diagram these dependencies among the packages. Then, we can use the connector to link the packages as in Figure 7.6.

What is Allowed on A Class Diagram ?

1. More than one package is possible in a class diagram.
2. It is possible that a class is represented in more than one package. For example, a class from a package can be imported into another package. The name of the imported class would indicate its parent package, as in Figure 7.6.

Figure 7.6 Cross-package class



What is Next ?

We have covered all the diagrams and their functions in the Elixir CASE tool. Now is the time to apply what we have learned, and that's what we shall do in the next chapter. In the next chapter, we will see how we can incorporate the LORE method and the Elixir CASE tool to develop a sample Library System.

Chapter 8

The Library System

The Library System presented here serves as an example of how a complete system is documented using the Elixir CASE tool. The example applies the LORE method to develop the Library System and shows how the Elixir CASE tool is used to facilitate the development and to enhance the system.

Analysis

The objective of the analysis phase is to model the real world interactions in the Library System. An analysis team would have been formed to identify the various objects that interact with, or are part of, the system.

The analysis process consists of the following 3 phases :

- I. Object Identification phase
- II. Behaviour Modelling phase
- III. Structure Modelling phase.

The techniques that are commonly used are :

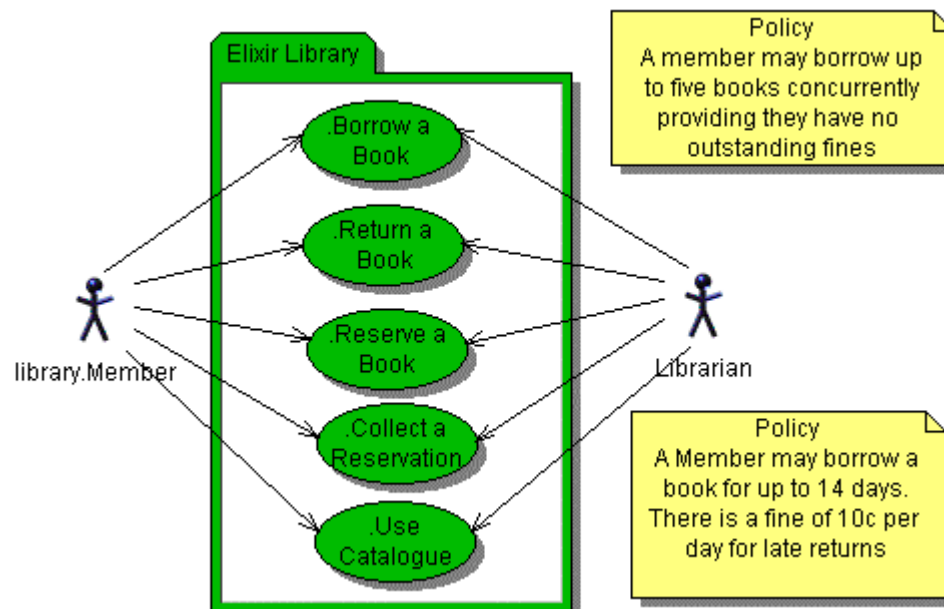
- ◆ Use Cases
- ◆ Noun Analysis
- ◆ Scenarios
- ◆ CRC (Class-Responsibility-Collaboration) Cards
- ◆ Patterns
- ◆ Data Analysis
- ◆ Sequence diagrams

We will be making use of some of these techniques in the Library system.

Object Identification Phase

Use Case Diagrams

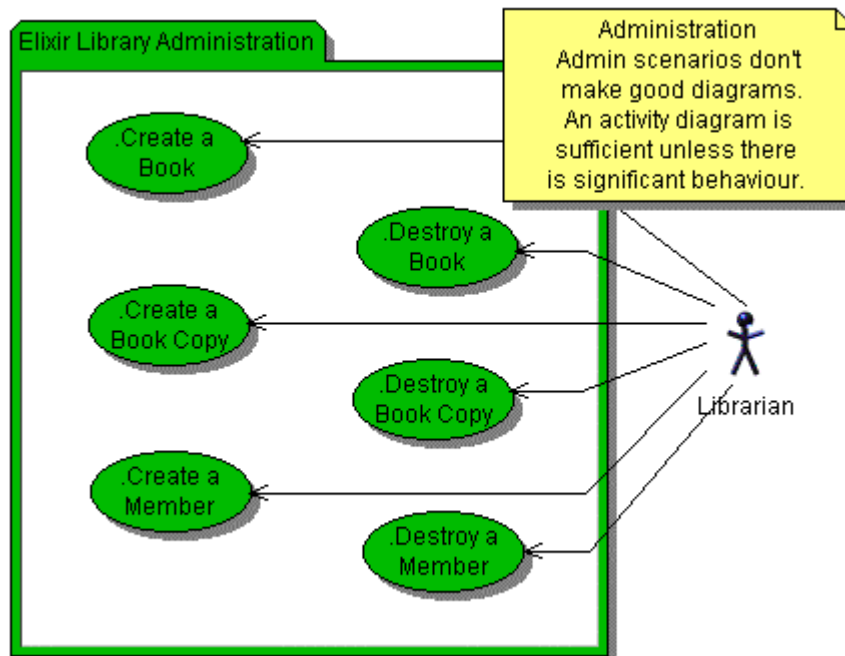
In the initial stages, the analysis team had identified the following use cases, which are documented in the use case diagrams below :

UCD1 Library System – Policy**Library Loan Policies**

1. Each member can borrow up to a maximum of 5 books at one time.
2. To borrow a book, a member must settle all outstanding fines.
3. A member may borrow a book for up to 14 days.
4. There is a fine of 10 cents per day per book for late returns, including Saturdays and Sundays.

Library Reservation Policies

1. Each member can reserve up to a maximum of 5 books at one time.
2. The reservation will be kept for a week, after which, if it is not collected, the reserved book will be assigned to the next person in line, if any, or put back on the shelf.

UCD2 Library System – Administration

The first use case diagram (UCD1) shows the basic essential functions of a library system and the interaction of the different roles with the functions. At the same time, the business policies that had been brought up are also documented.

The second use case diagram (UCD2) documents the administrative tasks required in the library system. These functions are of a different nature from the previous functions. Furthermore, being administrative in nature, they do not make good behaviour models. An early distinction introduced by the domain expert was that of Book and BookCopy. There may be many copies (BookCopy) of a single title (Book) within the library.

During the object identification phase, the team studied each use case to identify the collaboration and responsibilities of the different parts of the system. The objective of this study is to find all possible scenarios and objects attached to each use case. An effective way to do it is through role-playing.

If you are new to object identification, it is good practice to document the scenarios in textual form as it helps to identify the objects in the system. The following text fully describes the scenarios obtained from the study.

Textual Description of Basic Functions

Borrow a Book – Successful

The member brings a book and membership card to the counter and requests the librarian to process the loan. The librarian checks the member's records against the library loan policies for any violations. As no library loan policies have been violated, the librarian proceeds to create a loan, and records the due date of the loan. The librarian also proceeds to update the status of the book.

Borrow a Book - Exceeds Loan Limit

The member brings a book and membership card to the counter and requests the librarian to process the loan. The librarian checks the member's records against the library loan policies for any violations. The member has exceeded the loan limit. The loan process is not successful.

Borrow a Book – With Outstanding Fines

The member brings a book and membership card to the counter and requests the librarian to process the loan. The librarian checks the member's records against the library loan policies for any violations. The librarian finds the member has outstanding fines from previous late returns. The librarian then requests for settlement of the outstanding amount. If the member pays up, scenario “Borrow a Book – Successful” or “Borrow a Book – Exceeds Loan Limit” is called. Else, the loan process is cancelled / stopped.

Return a Book – On Time

The librarian takes a book from the returns counter and checks the due date of the loan. If the due date is earlier than or is the same as the current date, then the current date is recorded as the return date and the loan is expired normally. The librarian proceeds to update the status of the book to “available” and returns it to the shelf.

Return a Book – Overdue

The librarian takes a book from the returns counter and checks the due date of the loan. If the due date is later than the current date, then the fine is calculated based on the type of book and the number of days overdue. The fine is then added to the member's outstanding amount, and the loan terminated. The librarian proceeds to update the status of the book to “available” and returns it to the shelf.

Reserve a Book – Successful

The member asks the librarian to reserve a book, providing the book and membership information. The librarian locates the book, and checks the member's records against the library reservation policies for any violations. As no reservation policies have been violated, the librarian proceeds to record the reservation.

Reserve a Book – Exceeds Reservation Limit

The member asks the librarian to reserve a book, providing the book and membership information. The librarian locates the book, and checks the member's records against the library reservation policies for any violations. The member has exceeded the reservation limit. The reservation process is not successful.

Collect a Reservation - Successful

The member presents the “Book Available” notification letter and membership card to the librarian. The librarian locates the physical book and verifies the reservation. The reservation

is verified to be correct, and the “Borrow a Book – Successful” scenario is applied. The status of the reservation is then set to complete.

Collect a Reservation - Exceeds Loan Limit

The member presents the “Book Available” notification letter and membership card to the librarian. The librarian locates the physical book and verifies the reservation. The librarian then checks the member’s records against the library loan policies for any violations. The member has exceeded the loan limit. The “Borrow a Book – Exceeds Loan Limit” scenario is applied. The loan process is not successful.

Collect a Reservation – With Outstanding Fine

The member presents the “Book Available” notification letter and membership card to the librarian. The librarian locates the physical book and verifies the reservation. The librarian then checks the member’s records against the library loan policies for any violations. The Librarian finds the member has outstanding fines from previous late returns. The librarian then requests for settlement of the outstanding amount. If the member pays up, scenario “Borrow a Book – Successful” or “Borrow a Book – Exceeds Loan Limit” is called. Else, the loan process is cancelled / stopped and the reservation is cancelled.

Collect a Reservation – Incorrect Member Identity

The member presents the “Book Available” notification letter and membership card to the librarian. The librarian locates the physical book and verifies the reservation. The member is not the next in line for the book, so the librarian declines to let the book be borrowed. The status of the reservation record is not changed.

Notification of Reservation being Available for Collection

When a book copy is returned and the book has been reserved, the system will generate a “Book Available” notification letter. This letter will then be sent to the member.

Notification of Overdue Book

When a loan is overdue by more than 2 weeks, the system will generate a reminder letter. This letter will then be sent to the member.

Textual Description of Administration

Create a Book

The librarian creates a new book given its title, author, ISBN and publisher.

Create a Book Copy

The librarian creates an instance of book copy. The first copy is given an arrival number of 0, subsequent copies of the same book are assigned consecutively higher arrival numbers.

Create a Member

The librarian creates a new member given a name. The new member has no outstanding fine.

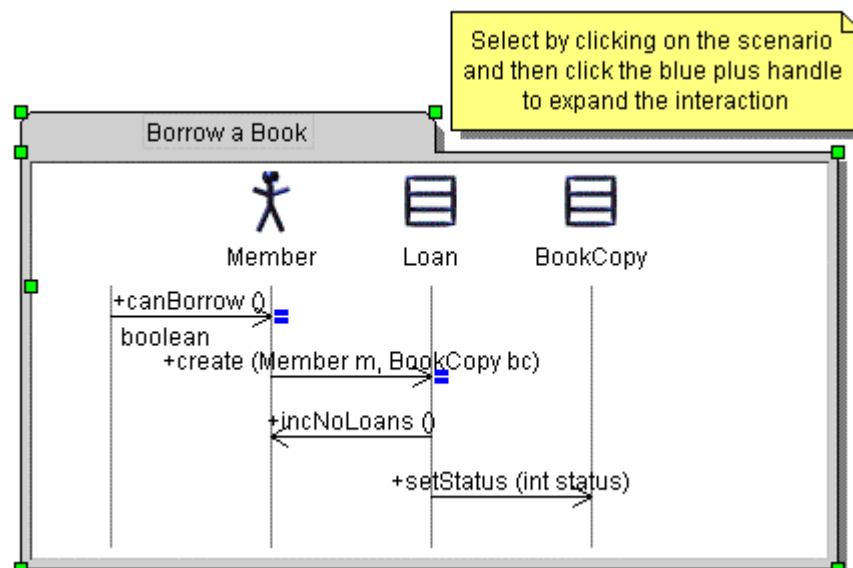
Noun Analysis

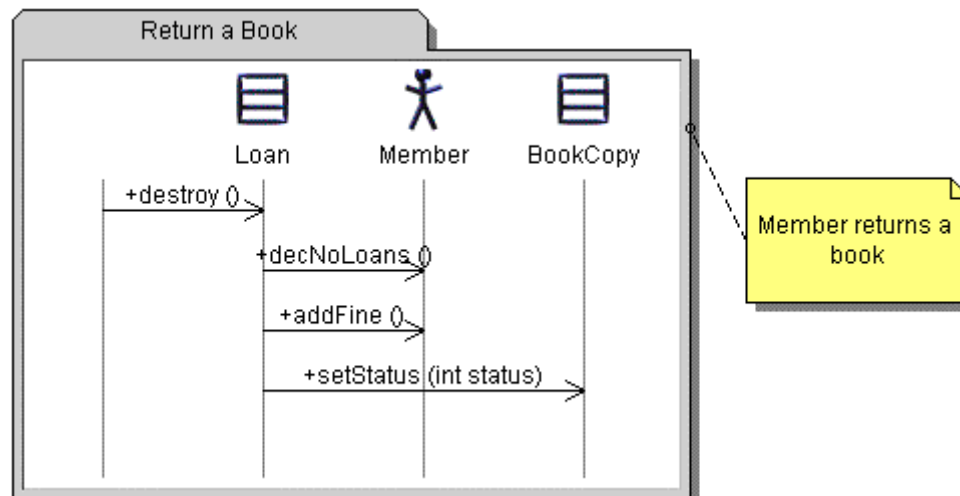
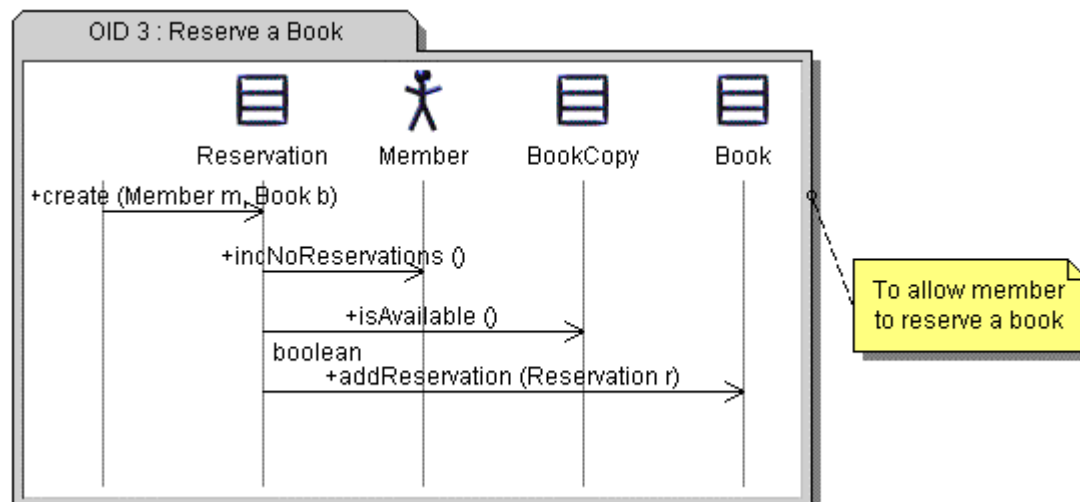
In the textual descriptions, the nouns have been underlined. We will therefore apply the noun analysis technique to identify the objects. Also, some nouns represent fields rather than objects; for example, "outstanding fine" is a field. Not all the nouns which have been underlined are useful in the object sequence diagrams, therefore we need to be selective. From our noun analysis exercise, a few objects have been identified to be useful in our study :

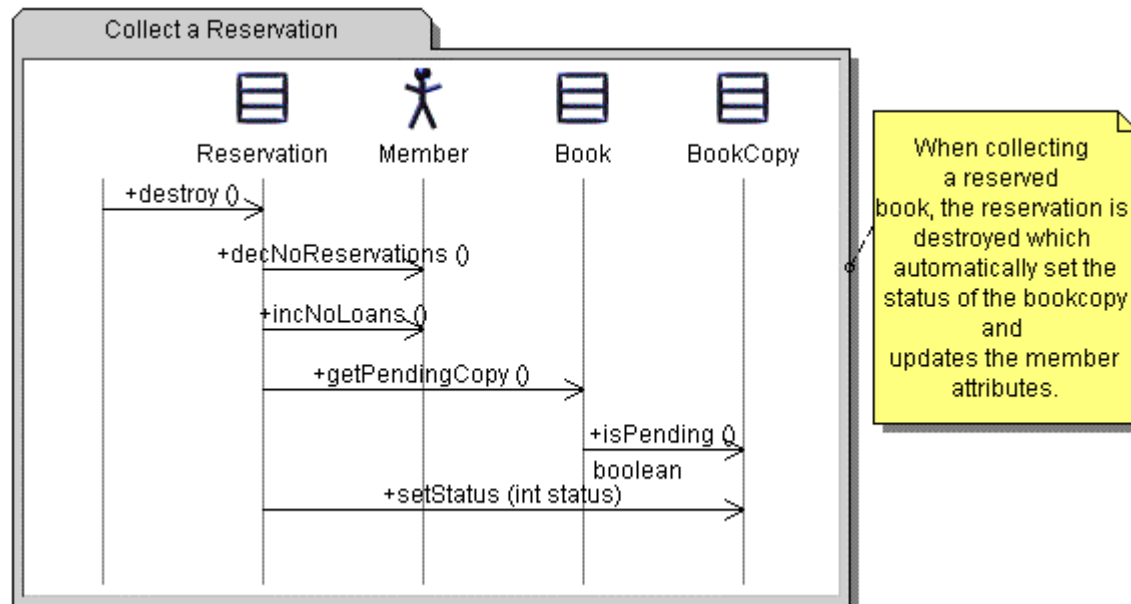
- ✓ Member
- ✓ Librarian
- ✓ Book
- ✓ BookCopy
- ✓ Loan
- ✓ Reservation

Behavioural Modelling Phase**Object Sequence diagrams**

After identifying the objects, we can create the object sequence diagrams base on these scenarios. They are shown below. As the scenarios are quite similar, we only show some of their object sequence diagrams.

OID1 Borrow a Book – Successful

OID2 Return a Book – Overdue*OID3 Reserve a Book – Successful*

OID4 Collect a Reservation - Successful**Pre-conditions and Post-conditions**

Upon completion of the OIDs, the next step is to assess possible pre and post conditions.

OID1 Borrow a Book

Pre-conditions : BookCopy status is "Available".
 Num_of_Loans (member) < Loan Limit (= 5)
 Post-condition : BookCopy status is "Borrowed".

OID2 Return a Book

Pre-condition : BookCopy status is "Borrowed".
 Post-condition : BookCopy status is "Available".

OID3 Reserve a Book

Pre-condition : Num_of_Reservations (Member) < Reservation Limit (= 3)
 Post-condition : Num_of_Reservations (Member) = Num_of_Reservations (Member) + 1

OID4 Collect a Reservation

Pre-conditions : BookCopy status is "Pending".

Post-conditions :

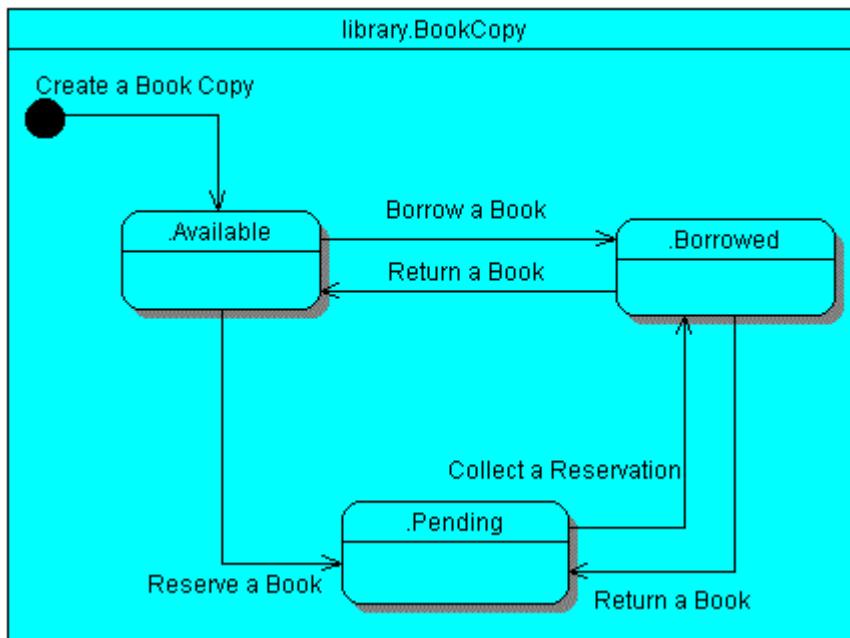
- Num_of_Loans (member) < Loan Limit (= 5)
- BookCopy status is "Borrowed".
- Num_of_Loans (Member) = Num_of_Loans (Member) + 1
- Num_of_Reservations (Member) = Num_of_Reservations (Member) – 1

Upon listing the pre and post conditions for each scenario, we discovered that there was no scenario to take a book into the Pending state, which is required in OID4. Actually, there are two situations which can result in a Pending BookCopy. In OID2, when a book is returned, it is kept for the next member on the reservation list, if any. In OID3, in the case of phone reservation, the book could already be Available. Therefore, upon reservation of the book, its status should be changed to "Pending".

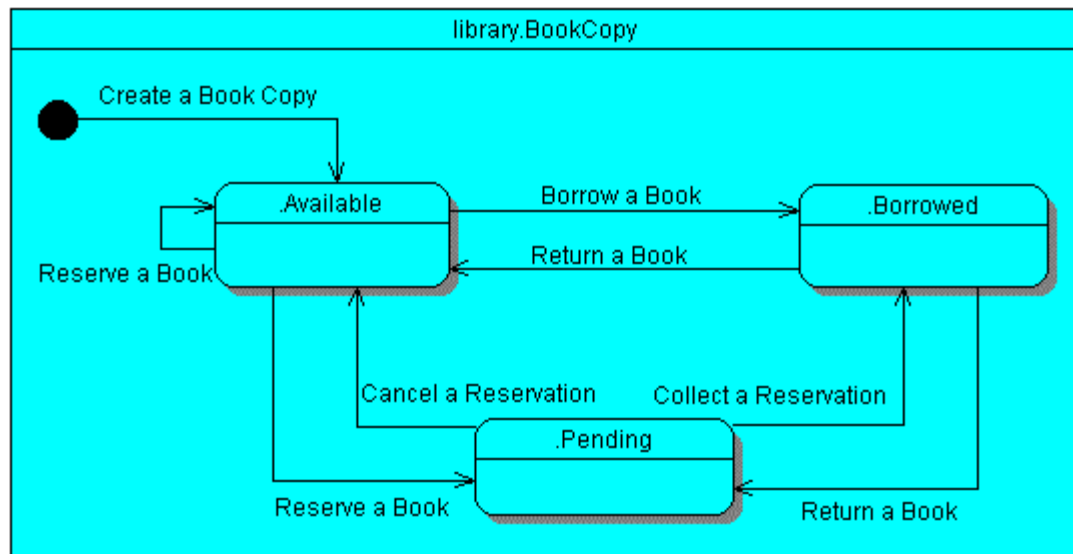
Statechart diagrams

Statechart diagrams are then produced to validate the pre and post conditions. For our example, we choose to produce a statechart diagram of BookCopy as it is an important class in the library system.

STD1 Initial BookCopy Statechart diagram

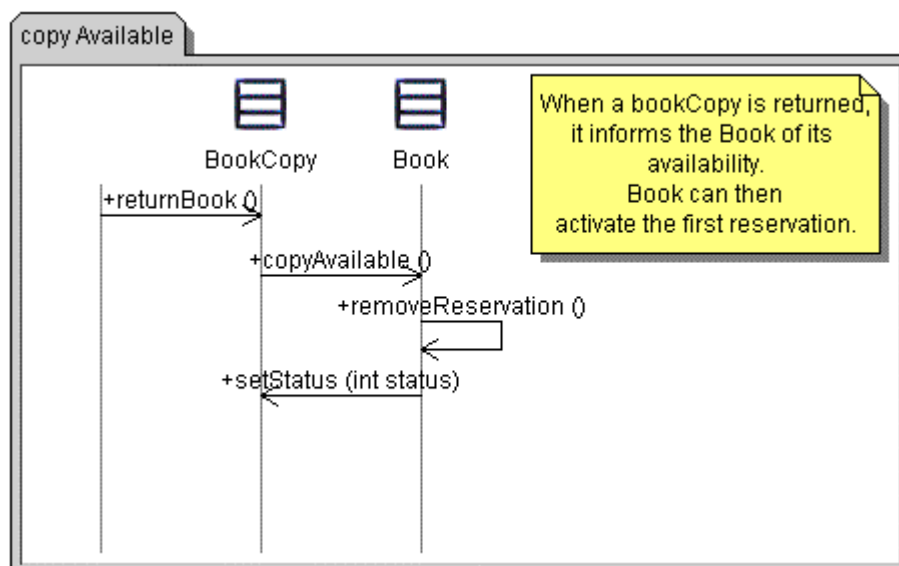


The Pending State has already been incorporated into our diagram, with a transition from Available state to Pending state, as discussed above. From the diagram, we managed to find further missing transitions. While checking for possible transitions from each and every of the states, we realized that there should be a transition from Pending state to Available state. Therefore, a "Cancel Reservation" transition is added to reflect a more accurate picture of BookCopy.

STD2 Revised BookCopy Statechart diagram**More Object Sequence diagrams and Statechart diagrams after Cyclical Analysis**

As a result of the changes, we will need to, firstly, make changes to our “Return a Book” scenario and, secondly, to add a new “Cancel a Reservation” scenario. However, instead of updating our existing OID2 : Return a Book, we choose to provide an extension scenario, Copy Available. It starts at the point where OID2 finishes. [See OID5].

OID 5 shows the CopyAvailable scenario. This scenario shows a message to self - `removeReservation`. It is an internal action not directly triggered by an external agent. We choose to represent it to make the add – remove cycle symmetrical.

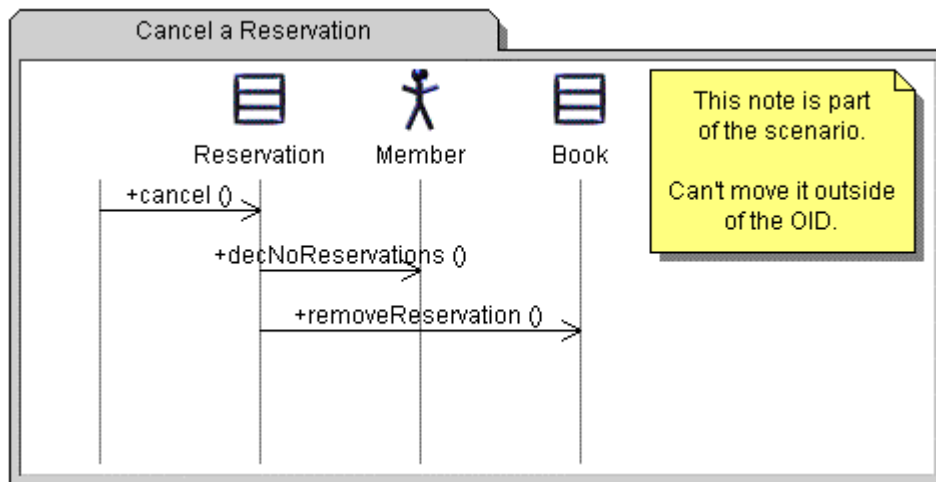
OID5 Copy Available

A new scenario “Cancel a Reservation” is added to provide a complete picture of the library system. The textual description of this scenario is as below :

Cancel A Reservation

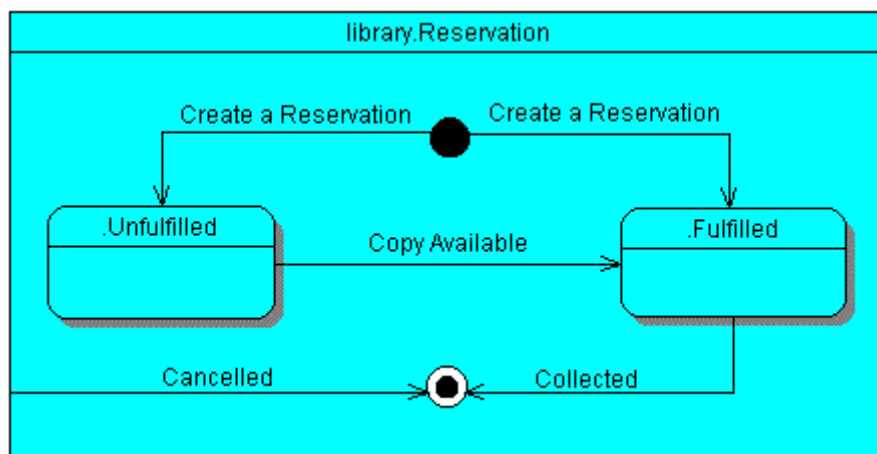
The member asks the librarian to cancel a reservation, providing the book information and membership information. The librarian locates the reservation and removes it. If the reservation cannot be found, no action needs to be taken.

OID6 Cancel a Reservation



As a final validation, a statechart diagram for the Reservation class has been created. Reservations is also a major class in the library system.

STD Reservation Statechart diagram



In the Reservation statechart diagram, the transition from the Reservation class (container) to the Stop state illustrates a powerful feature of UML. The transition is inherited by all sub

states, namely, Unfulfilled and Fulfilled, and hence saves having to clutter the diagram with additional transitions.

Structure Modelling Phase

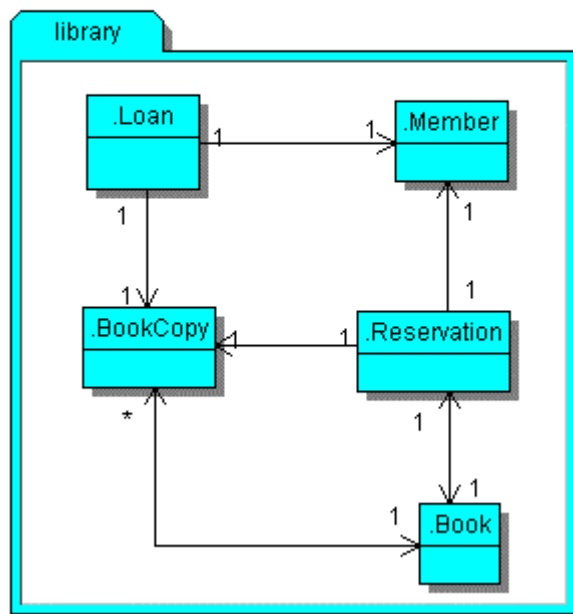
Class Diagrams

The last step of the analysis phase is to produce the class diagram. We first obtain a first cut diagram by transformation from the OIDs that we have previously drawn.

We first draw all the classes that have been used in the OIDs. Then, we put in the links in the OIDs. For example, in OID 1, Loan calls Member and BookCopy. This relationship is also drawn in the Class Diagram as shown. We then proceed to OID 2 to do likewise. This process is repeated until all the OIDs have been traversed. Thus, we obtain our first cut class diagram.

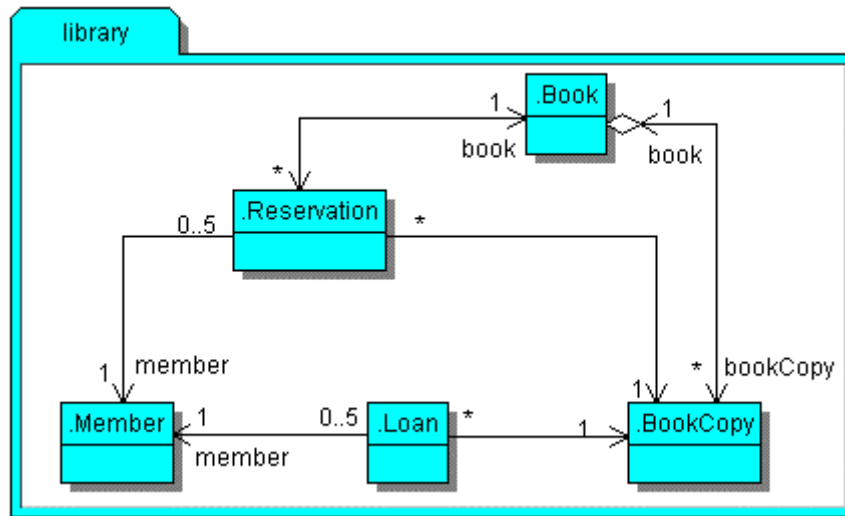
CD1 First-cut Class Diagram

This is the first cut class diagram, with information inferred from the interaction diagrams.



Upon further analysis, we obtain the final Class Diagram as shown below.

CD2 *Final Class Diagram*



Exercise

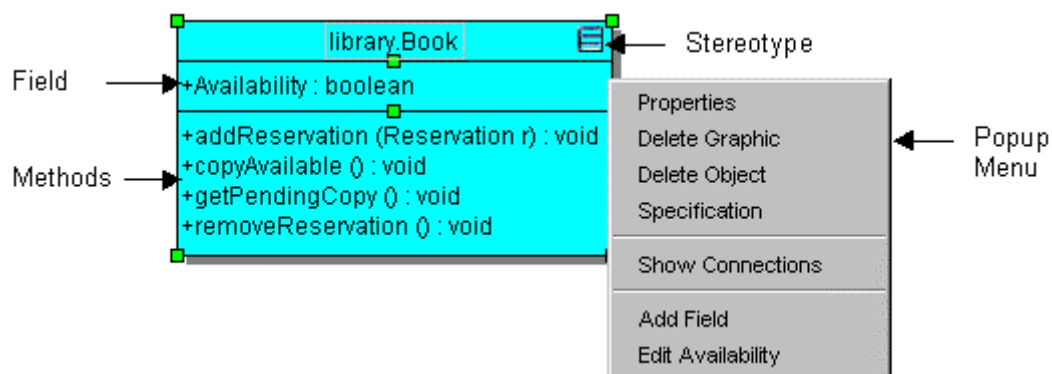
As an exercise, you may want to model library branches, where a library has many members and many branches, and each branch has many books.

APPENDIX A

Popup Menus

When you right click on any object you have created, a menu of commands will pop up, as shown in Figure A.1. This set of commands may differ for each object, as a command may be appropriate for some objects but not for others. In fact, clicking on different sections of a class will also call up different sets of commands, as we will see later. In this section, we will look at these commands and what they do.

Figure A.1 Popup Menu for a Class



Popup Menu Items

The following is a list of popup menu items you can expect from the tool.

Properties

To call up the Properties window which pertains to the particular object. There are various tab pages in the Properties window. Their functions are described in Appendix B.

Delete Graphic

To delete the graphical representation of the object model from the diagram page, but with the object model still existing in the database. Graphics of the links from or to the deleted object will also be deleted from only the diagram page. You can still repaint the object graphic again by using <Ctrl>-Click on the corresponding object type in the Toolbox.

However, there are exceptions. The Bookmark, Image, Note and Shortcut objects are graphics only, and they have no object models. Therefore, a Delete Graphic command will

remove them completely from the database. You can still recover them by Undo-ing, but not after a commit is done.

Delete Object

To delete the object model from the diagram page as well as the database. The physical links connecting the deleted object model to other object models will also be deleted from the database.

In the case of the Diagram object, the link to the page that it points to is lost, but the object models that were drawn on the page still exist in the database. You can verify this from the Specification Editor.

Specification

To bring you to the Specification Editor. The Specification Editor is described in Chapter 3.

Show Connectors

Sometimes, your diagram may have too many connections which makes it look really complicated. Then, you may like to temporarily remove the links from the diagram (using Delete Graphic). Subsequently, you may like to concentrate on just a few of the objects, and therefore, would need to know the links between these objects. Show Connection thus allows you to reinstate specific links.

To do so, you will need to position your mouse on the object whose links you would like to see. Then, right-click on it to display the pop-up commands. Upon choosing Show Connections, all the connections from or to the object will be shown.

Show Contents

This command is only valid for Activity, Scenario, Class (container) and Package objects in the Toolbox. These are the Container objects. Previously, the children objects in these containers may be hidden; **Show Contents** makes all these object graphics appear again. However, the links are not shown. To do so, use **Show Connections**.

Tip :	<i>This can also be used to auto-resize a container object to the optimal size and to auto-arrange the object graphics in the container.</i>
--------------	--

Add Node

This command is used by the connectors only. Nodes refers to the green handles on the connector. This command is used to add nodes on the connector so that it can be manipulated (by pulling the green handles) to bend the line as is required.

Delete Node

This command is used by the connectors only. It is the reverse of Add Node, that is, it removes the node from the connector. To do so, select the node to be removed and choose the Delete Node command from the popup menu. When you delete a node, the connector will spring back into a straight line at the 2 adjacent nodes.

Add Field

This command will only appear if the mouse is clicked when the pointer is on the Field section of a class object. It will call up the Fields sub-Property window to allow you to add a new field to the class.

Edit *Fieldname*

On a class object, we can choose to make visible some or all the fields available in the class such that they are shown on the class object on the diagram page as in Figure A.1 [refer to Appendix B – **Show** tab page]. The “Edit *Fieldname*” command will appear when you click on a particular field. It will call up the Fields sub-Property window to allow you to make changes to the current field.

On Figure A.1, the mouse was clicked on the “Availability” field, and the “Edit Availability” command appears in the popup menu.

Add Method

This command will only appear if the mouse is clicked when the pointer is on the Methods section of a class object. It will call up the Methods sub-Property window to allow you to add a new method to the class.

Edit *Method*

On a class object, we can choose to make visible some or all the methods available in the class such that they are shown on the class object on the diagram page [refer to Appendix B – **Show** tab page]. The “Edit Method” command will appear when you click on a particular method. It will call up the Methods sub-Property window to allow you to make changes to the current method.

On Figure A.1, if the mouse pointer was clicked on the “addReservation” method, the “Edit addReservation” command appears in the popup menu.

A P P E N D I X B

Properties

Every object model has properties. Property refers to the characteristics of the object model. For example, a class has a name, methods, fields, and it may belong to a package.

To call up the properties of an object model, you can double click on the object graphic and the Properties window will appear on the screen. Alternatively, you can right-click on the object graphic and select Properties on the popup menu. On a Properties window, there will be one or more tab pages, depending on the object type. Each object model has its own set of property tab pages, and it may or may not differ from those of other object models.

In this section, we will look at the different properties an object model has and how to define these properties.

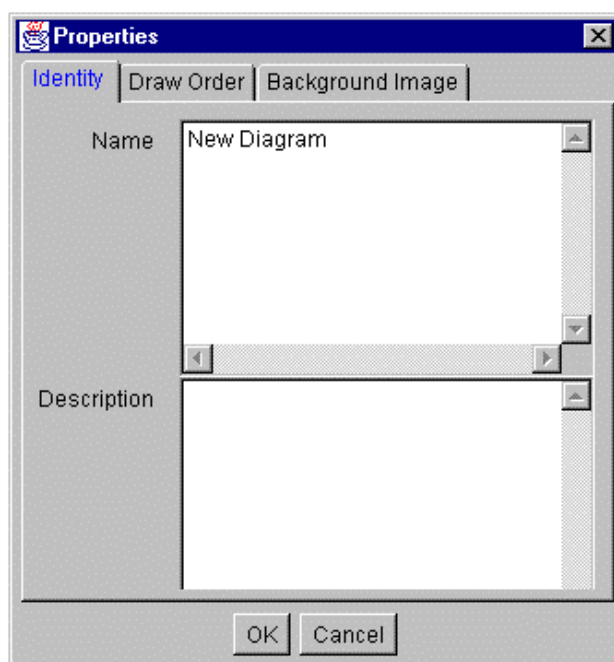
The Diagram Page

A diagram page has properties, as shown in Figure B.1.

Identity

The Identity tab page is available for all object models to name and describe the object model. There are variants of the Identity tab page, as we will see later under the section *Other Property Tab Pages*.

Figure B.1 Identity tab page



Draw Order

A diagram page consists of object graphics, and these graphics would have to be drawn on the screen one by one. Therefore, Draw Order refers to the order in which each object graphic is drawn. For example, in Figure B.2(i), Package A is drawn before Class A, and Class A before Class B. On a diagram, they will look as in Figure B.2(ii). So, the earlier it is drawn, the lower it will be in the list.

Figure B.2(i) Draw Order tab page

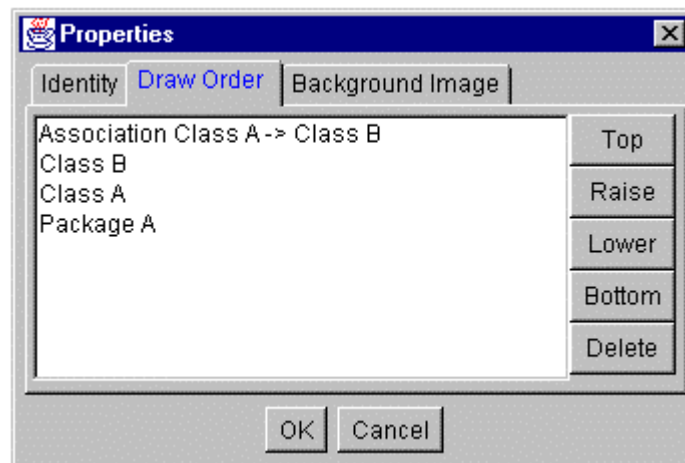
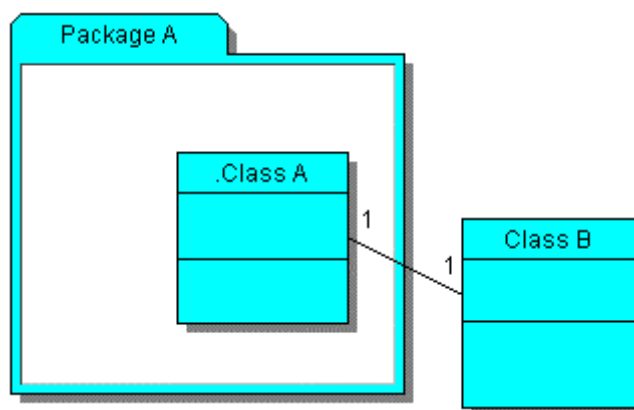


Figure B.2(ii) Corresponding Diagram , showing the object graphics on the page



The left section of the Draw Order tab page shows all the object graphics in their draw order. On the right side, there are 5 buttons. These are the operations to allow us to change the draw order of the objects. You need to select one object from the list, by highlighting it, and these operations will work on that selected object as described in the table below.

Operations	Action
Top	To place the object at the top of the list, that is, it will be on top of all objects.
Raise	To swap position of the object with the one above it so that it is now above the other object.
Lower	To swap position of the object with the one below it so that it will be below the other object.
Bottom	To place the object at the bottom of the list, that is, it will be below all the other objects.
Delete	To delete the object from the list so that it is no longer on the diagram but it is still in the database.

Knowing the draw order of objects on a page is useful in revealing object graphics that have been hidden under other object graphics.

Note

Changing the draw order of the object graphics in a diagram has no effect on the parent-child relationship that is already established when you first draw them.

Also, in Figure B.2(ii), Class B, being outside Package A, has no parent-child link to Package A. Moving Class B such that it is on top of Package A will not establish the link. To do so, we will need to re-parent Class B. Refer to Identity tab page (with Activity), Identity tab page (with Parent) and General tab page on how to re-parent.

Background Image

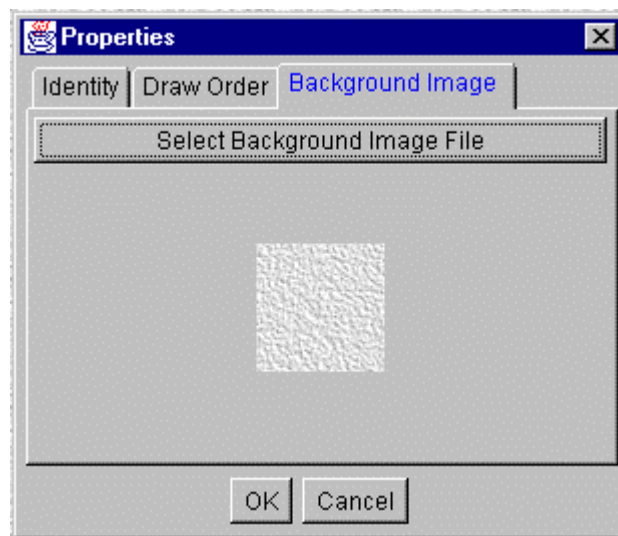
Figure B.3 shows the Background tab page. It allows you to select the background image that you want for your diagram.

After selection, the selected image will be shown to allow you to confirm your selection.

Only .gif and .jpg files can be used.

Tip : Using a larger image as background makes repainting of the screen faster.
--

Figure B.3 Background tab page



General Objects

This section covers the properties of objects created using a tool under the **General** section of the toolbox. All of these have properties except for *Bookmark*.

Diagram

A Diagram has 4 tab pages, as shown in Figure B.4(i). The first 3 have already been covered previously. Now we shall look at the User tab page.

User

The User tab page displays the list of user-defined properties for the object. Examples of user-defined properties are shown in Figure B.4(i) and B.4(ii) below. You can define your own set of properties (e.g. 'Expected Date of Completion').

Figure B.4(i) User tab page

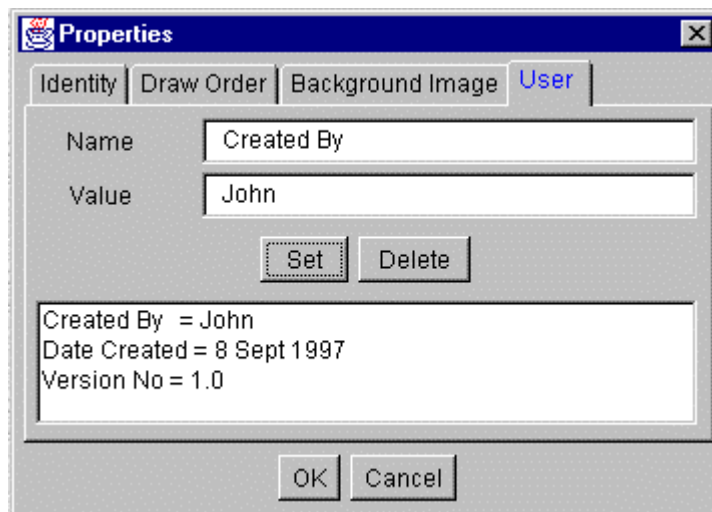
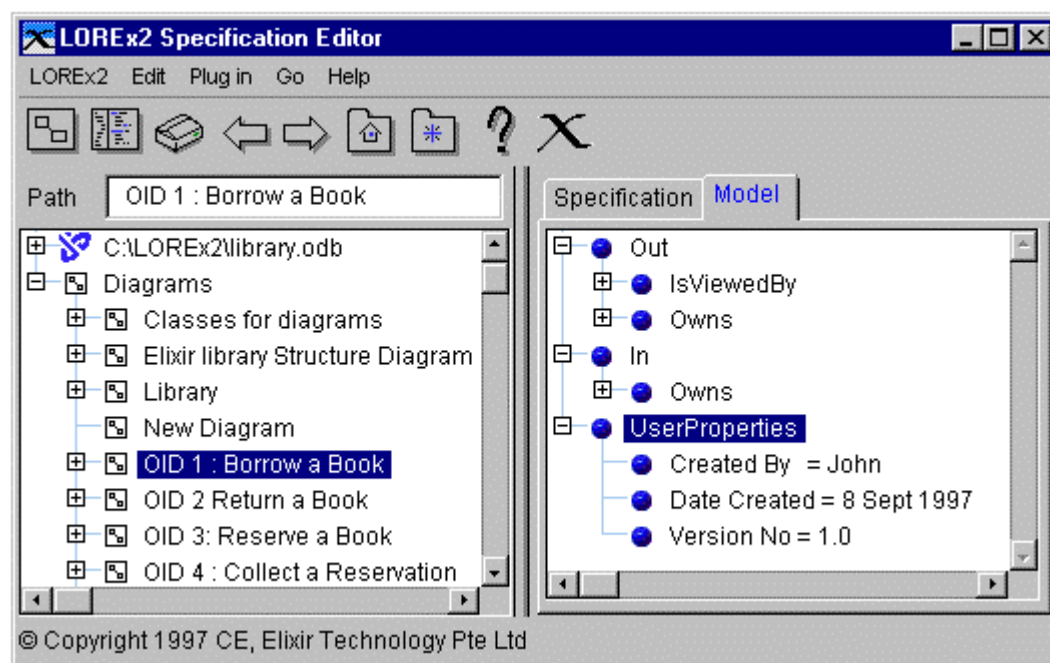


Figure B.4(ii) Corresponding UserProperties entry under Specification Editor



The fields are explained in the table :

Fields	Description
Name	This is a free text field where you can give a name to the identifier.
Value	This is also a free text field where you can put the value to be

	assigned to the identifier.
Set button	To assign Value to Name. The assignment will be shown on the empty area .
Delete Button	To delete the selected assignment.

Image

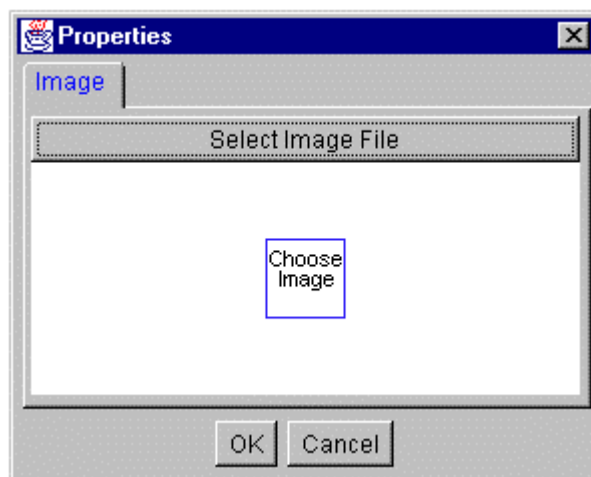
An Image object only has one tab page, the Image tab page.

Image

This tab page is similar to the one for background image [Figure B.3].

And similarly, only .gif or .jpg files can be used as an image.

Figure B.5 Image tab page for an Image object graphic



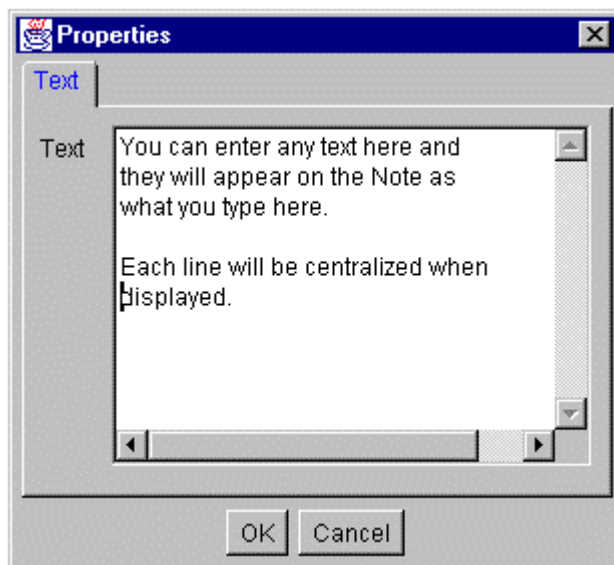
Note

A Note object also has only one tab page, the Note tab page.

Note

You can enter any text inside the Text area of a Note object, as shown in Figure B.6.

Figure B.6 Text tab page for a Note object graphic



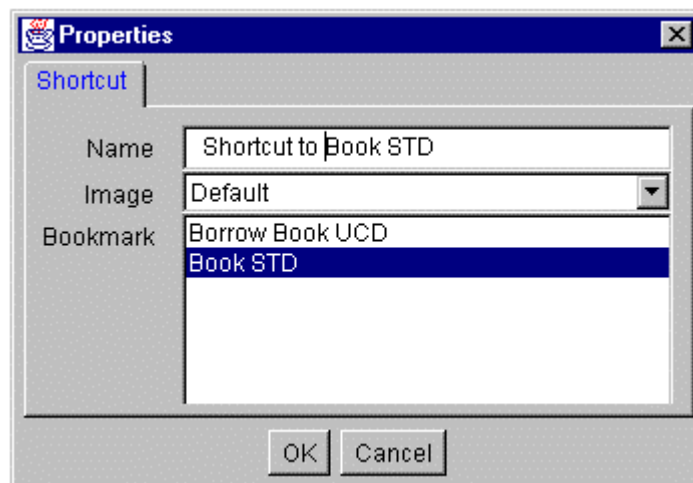
Shortcut

Similarly, the Shortcut object has only one tab page, the Shortcut tab page.

Shortcut

There may be a particular area on a diagram page which you access very often and from many places. So you would probably like to bookmark it. On the diagram page where you are currently working, you can add a Shortcut to this bookmark so that you can access the bookmark with just a click. The Shortcut tab page is as shown in Figure B.7. It has 3 fields, and they are described in the table below.

Figure B.7 Shortcut tab page for the Shortcut object graphic



Field	Description
Name	The name of the shortcut.
Image	You can choose your own icon in place of the default Shortcut icon.
Bookmark	Displays a list of all the available bookmarks across all Home pages. To create a shortcut, select one of the bookmarks and press the OK button.

Tip : You can also choose your own Shortcut icon. Add your own .gif or .jpg file in the Elixir CASE\shortcuts subdirectory and Elixir CASE will pick it up and include it as an option for selection under the field "Image".

Other Property Tab Pages

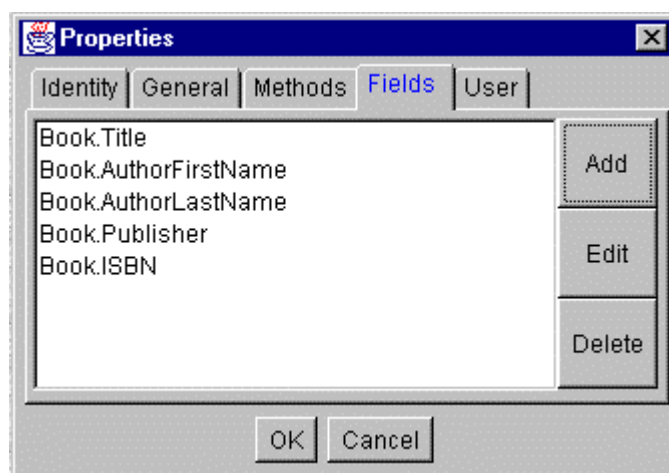
This section covers the property tab pages of all the other object types in the Toolbox. They are arranged in alphabetical order for easy reference.

Fields

Fields refer to the attributes of the class.

The Fields tab page can be found under the Properties of a Role, External System, Class, Class (Container) and Interface. On the left of the tab page are listed the fields of the current class. On the right side, there are 3 buttons to allow you to Add, Edit or Delete a method [refer to Figure B.8]. If you choose to add a new field or edit an existing field, you will be brought to another Properties window, a Sub-properties window for Fields (see Figure B.8).

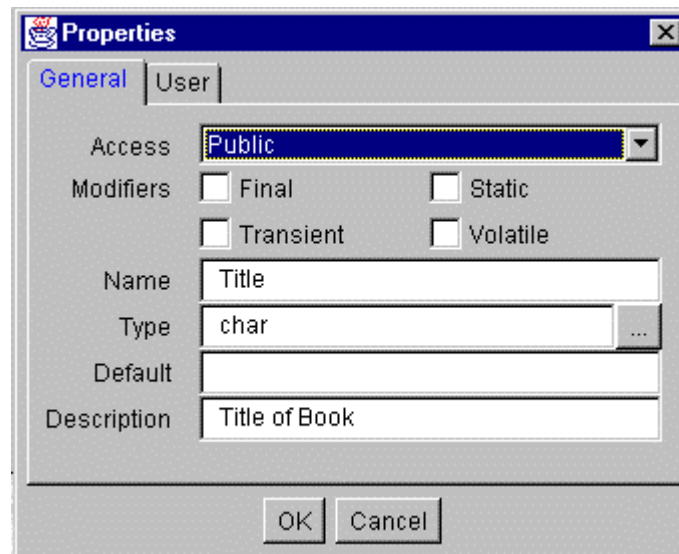
Figure B.8 Fields tab page



Fields.General

This tab page is to specify the characteristics of the Field, which are explained in the table below.

Figure B.9 Fields' Sub-Properties window upon choosing Add or Edit button

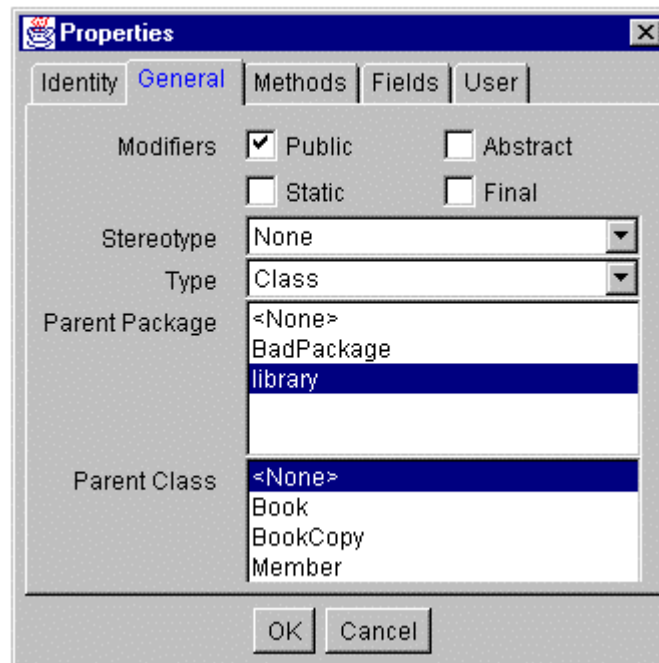


Field	Description
Access	Defines the access type of the field in a class.
Modifiers	These are field modifiers.
Name	Refers to the name of the field.
Type	Refers to the type of the field. If you click on the small box at the right, it will bring you to a dialog window to choose your types. Alternatively, you can key in the fully-qualified name of the class representing the type.
Default	Refers to default values. If not provided, this value will be automatically assigned.
Description	A free text description area.

General

The General tab page can be found under the Properties of a Role, External System, Class, Class (Container) and Interface. The fields on the page are explained in the table:

Figure B.10 General tab page



Field	Description
Access	Defines the access type of the class in a package.
Stereotype	Defines the stereotype of the class, if any.
Type	Defines whether it is a normal class, abstract class, final class, interface or data type.
Parent Package	A listing of all available packages in the database will be shown here. The highlighted package is the parent package of this class. The default is <None>. This field also allows you to re-parent the class [See Identity (with Parent) tab page].
Parent Class	Some classes may have another class as their parent, in addition to belonging to a package. A listing of all available classes in the selected package will be shown here. The highlighted class is the parent of the current class. The default is <None>.

Note

There are 2 other General tab pages which are called up by the Methods and Fields tab pages. They have a different set of functions which are specific to their respective calling tab pages. Therefore, we cover them under their respective sections.

Identity (with Activity)

As you can see in Figure B.11, there is an additional field “Activity” under this variant of the Identity tab page.

This activity field is applicable for use cases and scenarios. It is to allow a use case or scenario to be linked/relinked to an activity.

Figure B.11 Identity tab page (with Activity field)

The screenshot shows a 'Properties' dialog box with three tabs: 'Identity', 'Trigger', and 'User'. The 'Identity' tab is selected. It contains three main fields: 'Name', 'Description', and 'Activity'. The 'Name' field contains the text 'Borrow a Book'. The 'Description' field contains the text '"Borrow a Book" use case for "Elixir Library - Policy" Use Case Diagram'. The 'Activity' field is a list box with three options: 'None', 'Elixir Library - Administration', and 'Elixir Library - Policy', with the last option selected. At the bottom of the dialog are 'OK' and 'Cancel' buttons.

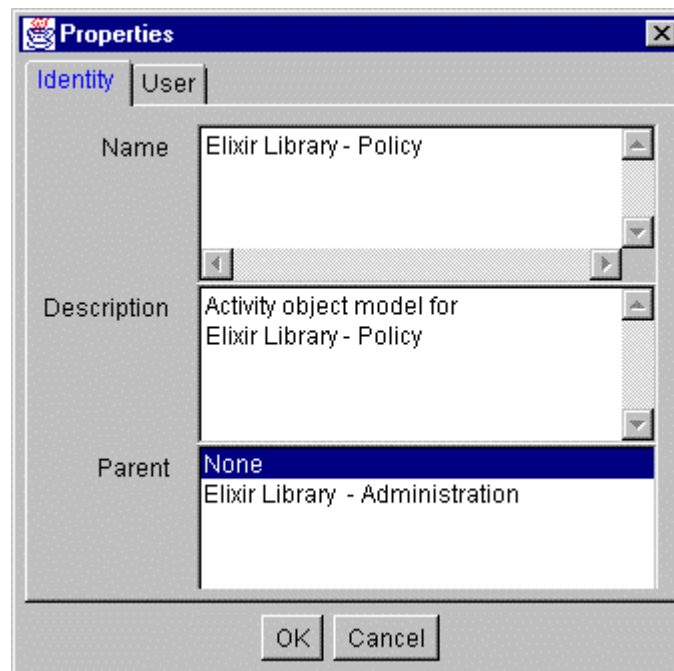
Field	Value
Name	Borrow a Book
Description	"Borrow a Book" use case for "Elixir Library - Policy" Use Case Diagram
Activity	Elixir Library - Policy

Identity (with Parent)

This is another variant of the Identity tab page, with an additional field “Parent”.

This Parent field is applicable for activities, the different state objects and packages. It allows reparenting of the object model.

Figure B.12 Identity tab page (with Parent field)



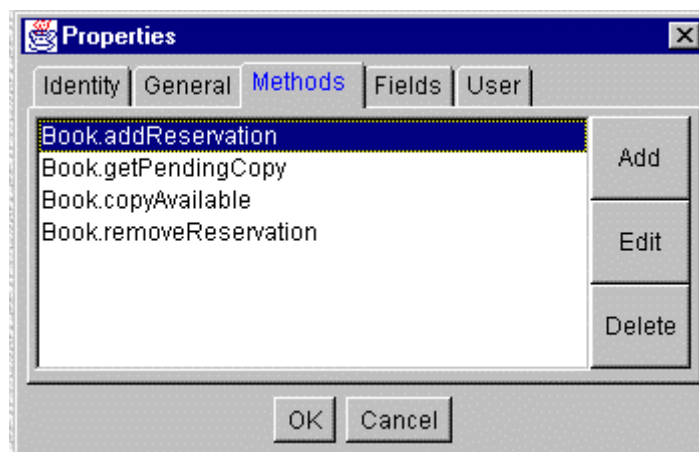
Interactions

Please refer to the section under “Trigger”.

Methods

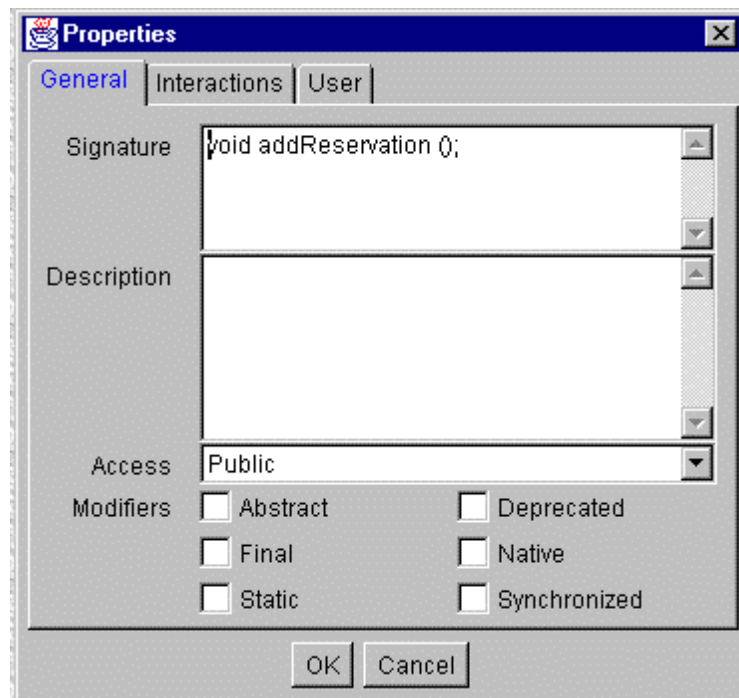
The Methods tab page can be found under the Properties of a Role, External System, Class, Class (Container) and Interface. On the left of the tab page will be listed the methods of the current class. On the right side, there are 3 buttons to allow you to Add, Edit or Delete a method [refer to Figure B.13]. If you choose to add a new method or edit an existing method, you will be brought to another window, the Properties window for method (see Figure B.14).

Figure B.13 Methods tab page



Methods.General

First, let us look at the Methods.General tab page, whose fields are explained in the table :

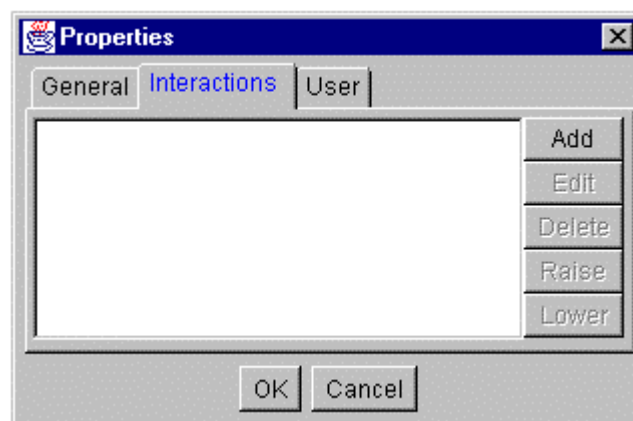
Figure B.14 Methods' Sub-properties window upon choosing Add or Edit button

Field	Description
Signature	Refers to prototypes, which includes expected arguments and return types.
Access	Defines the access type of the method in a class.
Modifiers	These are function modifiers.

Methods.Interactions

The Methods.Interactions tab page is used to invoke a message within the current method. It serves the same purpose as the Interactions tab page (Please refer to section under Triggers). The difference is that, instead of invoking the message from the Scenario, you can also invoke the message from the class method.

Figure B.15 Methods' Interaction tab page



Button	Description
Add	To invoke a new message.
Edit	To replace the selected message with another.
Delete	To remove the selected message.
Raise	To change the order of invocation of the selected message such that it is called before its current predecessor.
Lower	To change the order of invocation of the selected message such that it is called after its current successor.

Show

The Show tab page can be found under the Properties of a Class. It allows you to specify what you want to see on the class object on a diagram. The fields are explained in the table :

Figure B.16(i) Show tab page

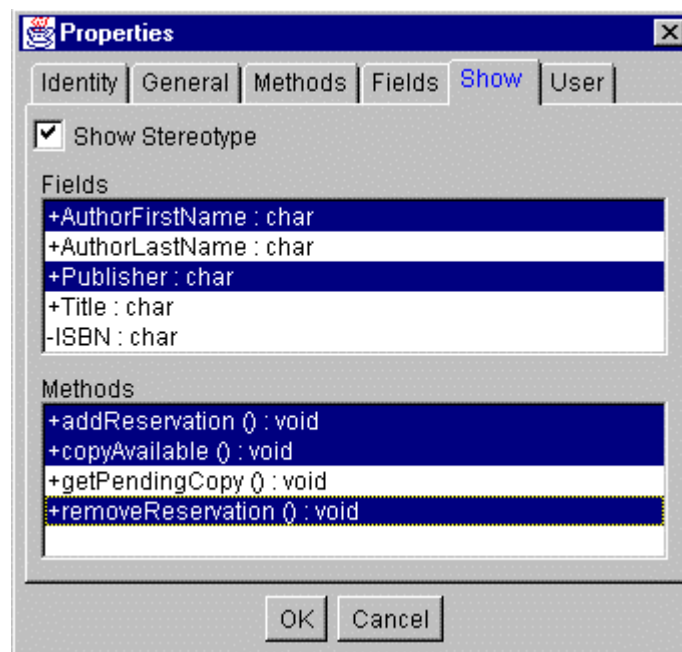
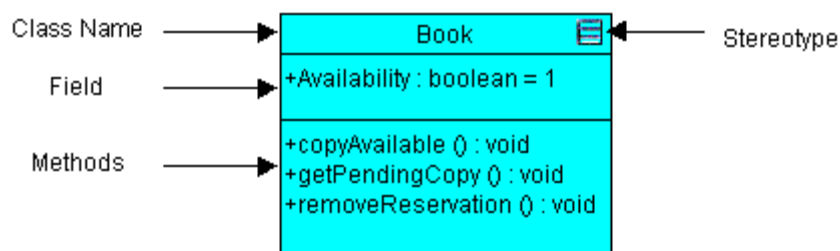


Figure B.16(ii) Class “Book” with Selected Fields and Methods and its Stereotype shown



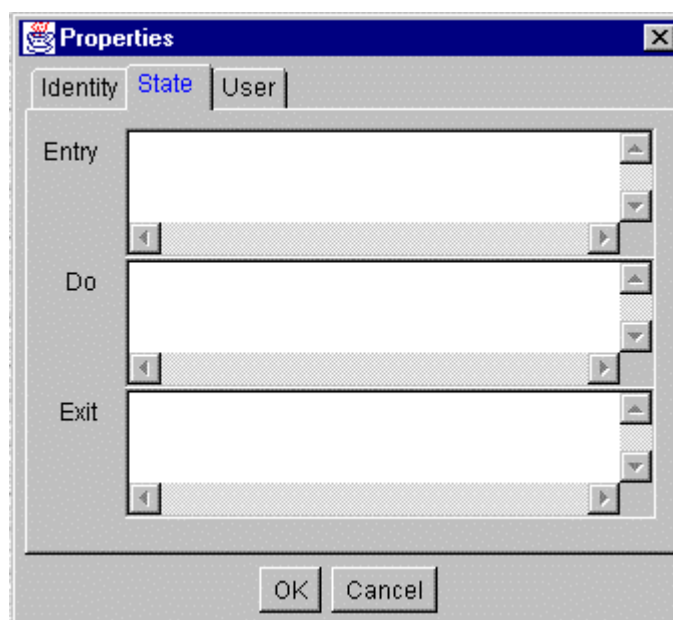
Field	Description
Show Stereotype	If the check box is selected, the stereotype of the class will be displayed at the top right hand corner of the class object.
Fields	The available fields are listed in this area. To display the fields on the class object, click on the field to highlight it. You can choose

	to select one or more fields for display.
Methods	The available methods are listed in this area. To display the methods on the class object, click on the method to highlight it. You can choose to select one or more methods for display.

State

The State tab page can be found under the Properties of a State. The fields are explained in the table :

Figure B.17 State tab page

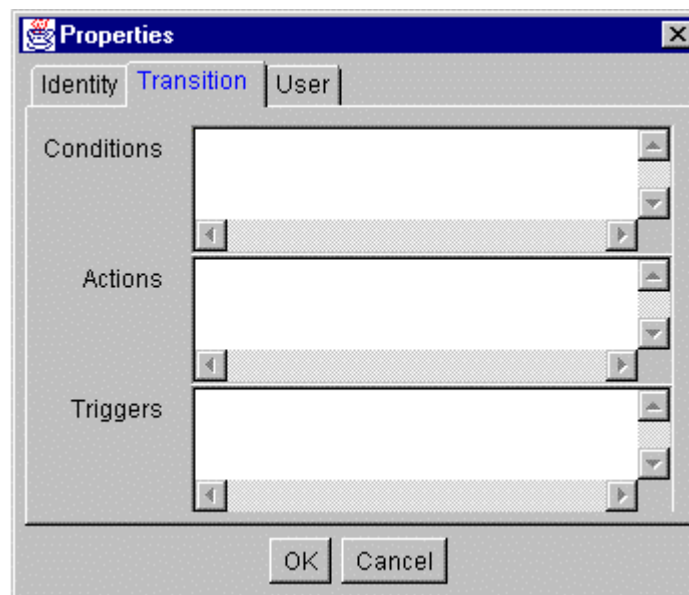


Field	Description
Entry	Allows the triggering of an event on all entries into a state. Syntax : entry/ EventName
Do	Allows an action to be performed continuously while in a state. Syntax : do/ EventName
Exit	Allows the triggering of an event on all exits out of a state. Syntax : exit/ EventName

Transition

The Transition tab page is found under the properties of the connector on a statechart diagram. It is used for documentation purpose to help us to easily map the statechart diagram to the object sequence diagrams. Its fields are explained in the table below.

Figure B.18 Transition tab page



Fields	Description
Conditions	The preconditions for this transition to occur.
Actions	The methods that are called during this transition.
Triggers	The events that are fired during this transition.

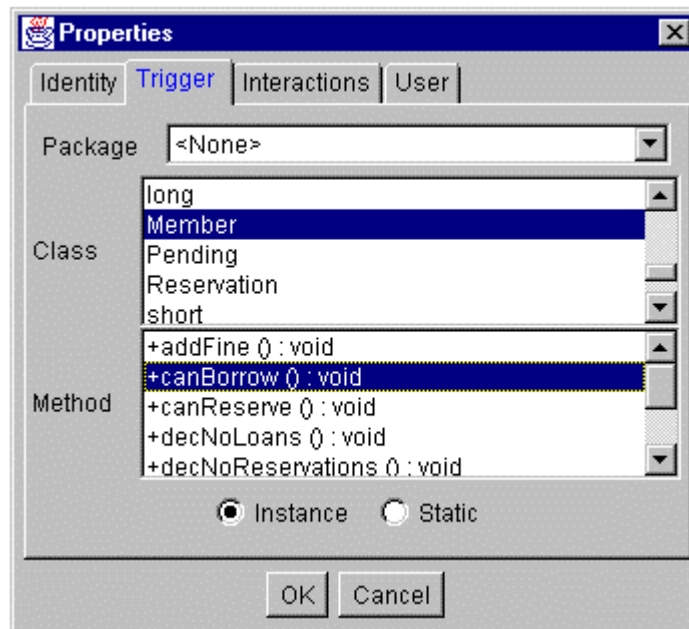
Trigger

A trigger refers to the first message sent to a scenario to start it off, and it is the first message received by the left-most object on a scenario. This trigger is specified in the Trigger tab page, as shown in Figure B.19.

The Trigger tab page can be found under the Properties of the Scenario and Use Case object types. For completeness in the system documentation, the Trigger tab page is provided for a use case to allow you to document the entry point of its corresponding scenario.

A top-down approach is to be adopted for this tab page, as explained in the table below.

Figure B.19 Trigger tab page



Field	Description
Package	A list of all the available packages.
Class	Upon choosing a package, all the classes in the package will be listed here.
Method	Upon choosing a class, all the available methods will be listed here.
Instance	Refers to instance functions, where the message is passed to the instance of the class.
Static	Refers to static methods, where the message is passed to the class, not the instance. A "\$" sign will be prefixed to a Static message. Instance and Static are, of course, mutually exclusive.

Interactions

All subsequent message invocations are to be through the Interactions or Methods.Interactions tab pages, show in Figure B.20(i) [Please refer also to the section under Methods.Interactions].

All the messages that have already been invoked on the current scenario will be listed in the Sender drop-down list box. Upon selection of one of the messages in the Sender field, if there already exists messages that are sent from within this message, they will be listed in the box. You can choose to add a new invocation or to edit / delete an existing message. You can also change the order of the invocations. You can perform these various tasks with the help of the pushbuttons, which are described in the table below.

Figure B.20(i) Interactions tab page

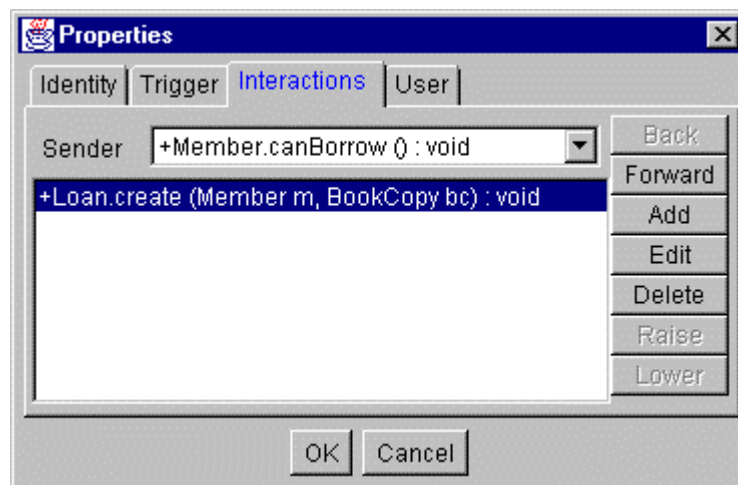
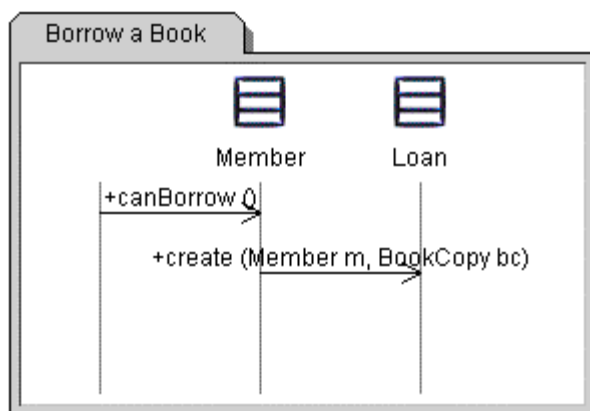
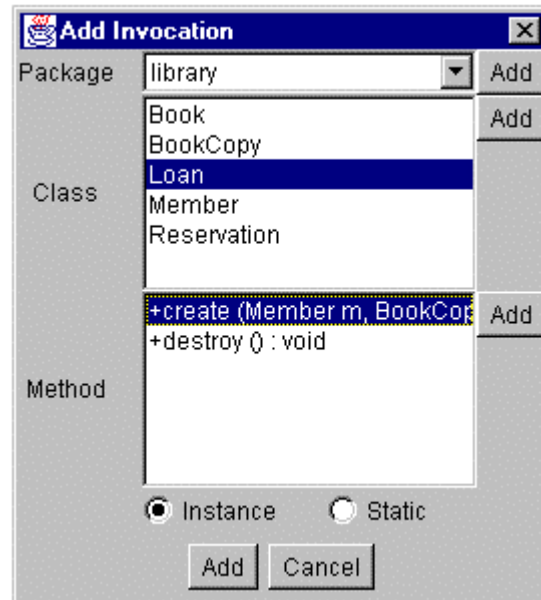


Figure B.20(ii) Corresponding Object Sequence diagram



Button	Description
Back	To move up one level of invocation.
Forward	To move forward one level of invocation.
Add	To invoke a new message. This will call up the Add Invocation window, as shown in Figure B.21.
Edit	To replace the selected message with another.
Delete	To remove the selected message. The lower level messages that are invoked within the deleted messages will also be deleted.
Raise	To change the order of invocation of the selected message such that it is called before its current predecessor.
Lower	To change the order of invocation of the selected message such that it is called after its current successor.

Figure B.21 Add Invocation

The 3 **Add** Buttons at the right side call up the Package Properties window, the Class Properties window and the Class' Method Sub-Properties window, respectively.